EXHIBIT 19

PA 1175086

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

June 02, 2004

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE UNDER 35 USC 111.

APPLICATION NUMBER: ***60/469,558***
FILING DATE: ***May 09, 2003***

By Authority of the
COMMISSIONER OF PATENTS AND TRADEMARKS

T. LAWRENCE
Certifying Officer

PG-1

5-12-3 60469558.050903

Prov

Attorney Docket No.: SUN-P030095.PRO

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Provisional Patent Application

I hereby certify that this transmittal of the below described documents is being deposited with the United States Postal Service in an envelope bearing Express Mail Postage and an Express Mail label, with the below serial number, addressed to the Commissioner for Patents P.O. Box 1450 Alexandria, VA 22313-1450, on the below date of deposit.

Express Mail Label No.:	EV349875949US	Name of Person Making the Deposit:	Anthony Chou
Date of Deposit:	05/09/03	Signature of the Person Making the Deposit:	Anthony Chou

Inventor(s): Andrew G. Tucker, John T. Beck, James D. Carlson, David S. Comay, Andrew D. Gabriel, Ozgur C. Leonard and Daniel B. Price

Title: OPERATING SYSTEM VIRTUALIZATION

Mail Stop Provisional Patent Application
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450
Sir:

This is a request for filing a:
PROVISIONAL APPLICATION FOR PATENT
under 37 C.F.R. § 1.51(c)(1)(i).

1. The following comprises the information required by 37 C.F.R. § 1.51(c)(1):

2. The name(s) of the inventor(s) is/are (37 C.F.R. § 1.51(c)(1)(ii)):

1. Andrew First Name	G. Middle Initial or Name	Tucker Last Name
2. John First Name	T. Middle Initial or Name	Beck Last Name
3. James First Name	D. Middle Initial or Name	Carlson Last Name
4. David First Name	S. Middle Initial or Name	Comay Last Name
5. Andrew First Name	D. Middle Initial or Name	Gabriel Last Name
6. Ozgur First Name	C. Middle Initial or Name	Leonard Last Name
7. Daniel First Name	B. Middle Initial or Name	Price Last Name

60469558.050903

3. Residence address(es) of the inventor(s) s numbered above (37 C.F.R. § 1.51(c)(1)(iii)):

1._____

2._____

3._____

4._____

5._____

6._____

7._____

4. The title of the invention is (37 C.F.R. § 1.51(c)(1)(iv)):

OPERATING SYSTEM VIRTUALIZATION

5. Statement as to whether invention was made by an agency of the U.S. Government or under contract with an agency of the U.S. Government.
(37 C.F.R. § 1.51(c)(1)(viii)):

This invention was made by an agency of the United States Government, or under contract with an agency of the United States Government.

[X] No.

[ ] Yes.

The name of the U.S. Government agency and the Government contract number are:_____

Transmittal of a Provisional Patent Application
(Under 37 CFR §1.51(c)(1))

Transmitted herewith is the above identified patent application, including:
_X_ Specification, claims and abstract, totaling _182_ pages.
____ Formal drawings, totaling _____ pages.
____ Informal drawings, totaling _____ pages.
____ Declaration and Power of Attorney
____ Assignment(s)
____ Assignment Recordation Form (duplicate)
____ Other: _____

60469558.050903

PAYMENT OF FEES

The filing fee for this provisional application, as set in 37 C.F.R. § 1.16(k), is $160.00, for other than a small entity, and $80.00, for a small entity.

[ ] Applicant is a small entity.

The full fee due in connection with this communication is provided as follows:

1. Not enclosed

   [ ] No filing fee is to be paid at this time.

2. Enclosed

   [X] Filing fee

   [ ] Recording assignment

   [X] The Commissioner is hereby authorized to charge any additional fees associated with this communication or credit any overpayment to Deposit Account No.: 23-0085. A duplicate copy of this authorization is enclosed.

   [X] A check in the amount of $160.00

   [ ] Charge any fees required or credit any overpayments associated with this filing to Deposit Account No.: 23-0085.

This application is filed pursuant to 37 C.F.R. § 1.53 in the name of the above-identified inventor(s).

Please direct all correspondence concerning the above-identified application to the following address:

WAGNER, MURABITO & HAO LLP
Two North Market Street, Third Floor
San Jose, California 95113
(408) 938-9060

[X] This transmittal ends with this page.

Respectfully submitted,

Date: 5/9/2003

By: _____
Anthony C. Murabito
Reg. No. 35,295

Virtualization and Namespace Isolation in Solaris (PSARC/2002/174)

John Beck, David Comay, Ozgur Leonard, Daniel Price, and Andy Tucker
Solaris Kernel Technology

Andrew Gabriel
Solaris Networking Technology

Revision 1.5

April 8, 2003

6046955 8.050903

Sun Proprietary/Confidential: Internal Use Only

Contents

Chapter 1

Introduction

A growing number of customers are interested in improving the utilization of their computing resources through consolidation and aggregation. Consolidation is already common in mainframe environments, where technology to support running multiple applications and even operating systems on the same hardware has been in development since the late 1960's. Such technology is an important differentiator in the Unix server market, both at the low end (virtual web hosting) and high end (traditional server consolidation).

This project introduces Solaris *zones*. Zones provide a means of virtualizing operating system services, allowing one or more processes to run in isolation from other activity on the system. This isolation prevents processes running within a given zone from monitoring or affecting processes running in other zones. A zone is a "sandbox" within which one or more applications can run without affecting or interacting with the rest of the system. It also provides an abstraction layer that separates applications from physical attributes of the machine on which they are deployed, such as physical device paths and network interface names.

Zones provide a number of features:

Security Network services can be run in a zone, limiting the damage possible in the event of a security violation. An intruder who successfully exploits a security hole in software running within a zone is limited to the restricted set of actions possible within that zone. For example, an application running within most zones cannot load customized kernel modules, modify kernel memory or create device nodes.

Isolation Zones allow the deployment of multiple applications on the same machine, even where those applications operate in different trust domains, require exclusive access to a global resource, or present difficulties with global configurations. For example, multiple applications running in different zones (but on the same system) can bind to the same network port using the distinct IP addresses associated with each zone. The applications are also prevented from monitoring or intercepting each other's network traffic.

Virtualization Zones provide a virtualized environment that can hide such details as physical devices and the system's primary IP address and host name from the application. This can be useful to support rapid deployment (and redeployment) of applications, since the same application environment can be maintained on different physical machines. The virtualized environment can be rich enough to allow separate administration of each zone; one could "give out the root password" to a zone administrator, knowing that any actions taken by that administrator would not affect the rest of the system. Such a facility is of interest to service providers who are looking for more granular ways to subdivide systems among customers (both internal and external).

Granularity Unlike physical partitioning technologies (such as domains or LPARs), zones can provide isolation at almost arbitrary granularity. A zone does not require a dedicated CPU, physical device, or chunk of physical memory; those resources can either be multiplexed across a number of zones running within a single domain or system, or allocated on a per-zone basis using the resource management features available in the operating system [33]. The result is that even a small uniprocessor system can support a number of zones running simultaneously; the primary restriction (aside from the performance requirements of the applications running within each zone) is the disk space to hold the files that are unique within each zone.

Transparency One of the basic design principles of zones is to avoid changing the environment in which applications are executing, except where necessary to achieve the goals of security and isolation. Zones do not present a new API or ABI to which applications must be "ported"; instead, they provide the standard Solaris interfaces and application environment, with some restrictions. The restrictions primarily affect applications attempting to perform privileged operations, as discussed in Chapter 5.

1.1 Zone Basics

Figure 1.1 shows a system with four zones. Zones 1, 2 and 3 are each running a disjoint workload in a sample consolidated environment. This example demonstrates that different versions of the same application may be run without negative consequence in different zones, to match the consolidation requirements. Each zone can provide an almost arbitrarily rich and customized set of services.

Basic process isolation is also demonstrated. Each zone is given access to a logical network interface; applications running in distinct zones cannot observe the network traffic of the other, even though their respective streams of packets travel through the same physical interface. Finally, each zone is provided a portion of the file system hierarchy. Because each zone is confined to its subtree of the file system hierarchy, the workloads cannot access each other's on-disk data.

Zone 0, enclosing zones 1-3, is the *global* zone. Processes running in this zone have (by and large) the same set of privileges available on a Solaris system today — they may load

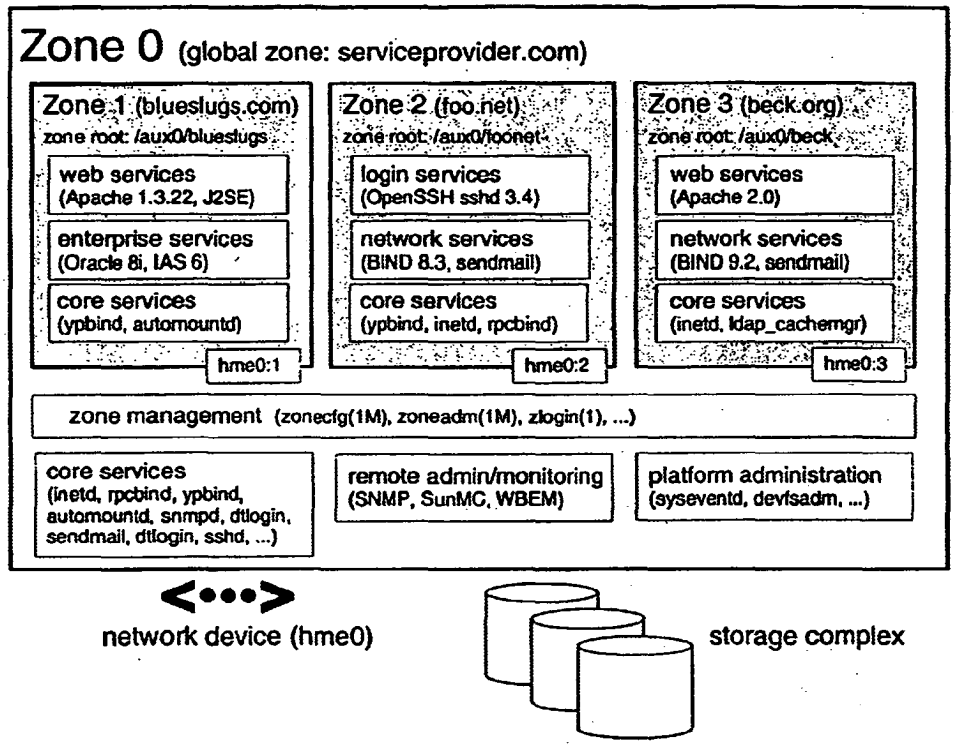

Figure 1.1: Server Consolidation Using Zones

kernel modules, access physical devices, etc.. An administrator logged into the the global zone can monitor and control the system as a whole (including the physical devices and network interface that are hidden from the other zones). The global zone always exists, and acts as the "default" zone in which all processes run if no zones have been explicitly created by the administrator.

Throughout this text, the term *global administrator* is used to denote a user with administrative privileges in the global zone. This user is assumed to have complete control of and responsibility for the physical hardware comprising the system, and of the operating system instance. The term *zone administrator* is used to denote a user with administrative privileges who is confined to the sandbox provided by a particular zone.

1.2 Zone Principles

This project required a number of decisions regarding isolation, visibility, and administrative complexity. The following list contains some of the basic principles that were used to guide these decisions, grouped according to general area.

1. Isolation

   - The kernel exports a number of distinct objects that can be associated with a particular zone. These include processes, file system mounts, network interfaces, and System V IPC objects.
   - No zone (other than the global zone) should be able to access objects belonging to another zone (including the global zone), either to control or modify those objects in some way, or to simply monitor or read them.
   - As much as possible, processes in a non-global zone should not be able to interfere with the execution of processes in other zones in the system. Although it is unlikely that this project can prevent all possible active denial-of-service attacks by privileged processes in non-global zones, accidental and relatively trivial interference should be prevented.
   - Appropriately privileged processes in the global zone can access objects associated with other zones. As much as possible, it should be possible to administratively and programmatically identify the zone with which each such object is associated.
   - Cross-zone communication may occur over the network (which is actually looped back inside IP, as with any traffic routed between logical interfaces in the same system), but not through other mechanisms without the participation of the global zone.

2. Compatibility

   - Applications in the global zone can run without modification, whether or not additional zones are configured.
   - With the exception of certain privilege limitations (discussed in Chapter 5) and a reduced object namespace, applications within a (non-global) zone can run unmodified.

3. Administration

   - Administration of the system infrastructure (physical devices, routing, DR, etc.) is only be possible in the global zone.
   - Administration of software services executing within a non-global zone is possible within the zone itself.

4. Security

   - Privileged processes in non-global zones are prevented from performing operations that can have system-wide impact, whether that impact would have performance, security, or availability consequences.
   - Even unprivileged processes in the global zone may be able to perform operations not allowed to privileged processes in a non-global zone. For example, ordinary users in the global zone are able to see information about every process in the system (although they may have to specifically request such a view). In environments where this is a significant concern, access to the global zone should be restricted.

5. Resource Management

   - Although this project provides administrative support for aligning zones with resource allocation boundaries, the features are treated as orthogonal from a design standpoint (i.e., such alignment is not mandatory). This is discussed further in Chapter 10.

1.3 Outline

The rest of this document is organized as follows. Chapter 2 describes some of the related efforts in this area both within and outside of Sun. Chapter 3 discusses how zones are created, logged into and destroyed, while Chapter 4 discusses system administration within zones, Chapter 5 describes security issues, including the privilege model for zones. Chapter 6 describes the changes and extensions to the process model. Chapter 7 describes file system changes, and Chapter 8 discusses networking issues. Chapter 9 discusses device issues, Chapter 10 addresses resource management and accounting, and Chapter 11 describes issues with various forms of inter-process communication. Chapter 12 describes how install, upgrade, and patch are addressed, and Chapter 13 discusses other projects within Sun and effects on third-party software. Finally, Appendix A contains tables of public interfaces introduced and modified by this project, Appendix B includes a copy of the XML DTD used for zone configuration, Appendix C contains draft man pages for new interfaces, and Appendix D contains draft man pages for modified interfaces.

6046958.050903

Sun Proprietary/Confidential: Internal Use Only

Chapter 2

Related Work

The idea of using operating system facilities to create secure application sandboxes has been around for quite a while; in Unix systems, the `chroot(2)` facility has been used for this purpose for many network service applications (most commonly in anonymous ftp servers). The chroot facility, however, only provides the ability to isolate processes in terms of file system access; it does not restrict access to other processes, network interfaces, or devices. In addition, a process with super-user privileges can easily escape the chroot restriction, as well as subverting any restrictions by creating device nodes and accessing physical devices.

FreeBSD took the idea a step further with the introduction of *jails* [21, 24], which provide a nearly complete process isolation facility. Like chroot, the use of the jail facility allows an administrator to restrict a process and its descendants to a specific part of the file system hierarchy, but also restricts network access to a specified IP address, restricts process access to processes within the same jail, and disables a number of privileges such as the ability to create device nodes or reboot the system. The result is that even a process with super-user privileges is unable to affect other processes when running inside a jail. Zones are based on the basic idea of jails, but extend the concept to provide a comprehensive facility that is integrated with core operating system services.

The Trusted Solaris Operating Environment [12] provides the ability to associate *labels* with a variety of system objects, including processes, files, and network interfaces. These labels can be used to restrict access, as well as to support *polyinstantiated* objects — distinct objects that have the same name but are distinguished by the label of the process that is viewing or modifying them. For example, the contents of `/etc/resolv.conf` might appear different depending on which process is reading the file, and one process' updates might not be seen by other processes depending on their respective different labels. Trusted Solaris also supports a multi-level hierarchy of labels, where processes with label A are able to access objects with the label A as well as any objects associated with labels that are "dominated" by label A in the label hierarchy. As is possible with jails and zones, labels can be used to create isolated containers for applications that execute on the same system. Although the label functionality is certainly more comprehensive and flexible than that provided by the other technologies, the administrative complexity of the model appears daunting to those

unfamiliar with it. Section 13.3 contains discussion of plans to leverage the zone functionality in future versions of Trusted Solaris.

Another "trusted" operating system, HP's Trusted Linux [6], uses an approach closer to jails (or zones) to meet the requirements of a secure OS. In this OS, *compartments* are defined to isolate different applications on the same system. The compartments represent a partitioning of objects within the system (including partitioning the file system hierarchy using a chroot-like approach). In addition, a set of access control rules can be used to control cross-compartment communication; for example, one can specify whether processes in one compartment are allowed to attach to System V IPC objects created by another compartment.

Another approach to virtualization is to virtualize the physical state of the machine, so that multiple operating system instances can simultaneously share the same physical hardware. This is the approach taken by *virtual machines*, which create a software model of the hardware state, and *logical partitions*, which use hardware support (such as the introduction of an additional privilege level) to isolate privileged programs such as operating system kernels. Such approaches have primarily been promoted by IBM and others in the mainframe arena, but have started to be applied to other systems as well [1, 2]. A project currently under development at Sun is working on providing the infrastructure needed to implement functionality similar to logical partitions for SPARC processors [9]. While such hardware-based virtualization can provide greater isolation than OS-based virtualization such as zones or jails, it does reduce the amount of sharing possible (e.g., text pages are not shared between partitions) and requires the administration of a number of independent operating system instances.

Chapter 3

Zone Administration

A freshly installed Solaris system will not have any (non-global) zones. The first step toward bringing up a zone is to create a configuration which specifies various parameters for that zone. Once a configuration is complete, it can be installed onto the system, then it can be booted, logged into, and administered. Then as needed it can be halted, rebooted, destroyed, etc. What each of these steps means and how each works are described in this chapter.

There are three new commands that are used for performing these steps: `zonecfg(1M)` to prepare the configuration, `zoneadm(1M)` to install and boot the configuration, and `zlogin(1)` to login to and administer the booted zone. In addition, the new command `zonename(1)` is useful for reporting the name of the current zone.

3.1 State Model

The various steps mentioned above can be thought of as transitions in a finite state machine. This section describes the state model that underlies the FSM abstraction. See Figure 3.1 for a graphical representation of this model.[1]

A zone can be in one of four states:

- The CONFIGURED state: a zone's configuration has been completely specified and committed to stable storage.
- The INSTALLED state: a zone's configuration has been laid out on the system: packages have been installed under the zone's root path.

[1]To keep the diagram from being too confusing, it appears that `zoneadm boot` is applicable only from the READY state, but this command is also applicable from the INSTALLED state; this would result in passing through the READY state on the way to the RUNNING state. Likewise, to keep the diagram simpler, it appears that a zone reboot, whether from a zone administrator issuing the `reboot` command or a system administrator issuing `zoneadm reboot`, results in a transition from the RUNNING state back to the same state, without going through any other intermediate states on the way. In reality, however, such a reboot results in a cycle through the INSTALLED and READY states before returning to RUNNING.

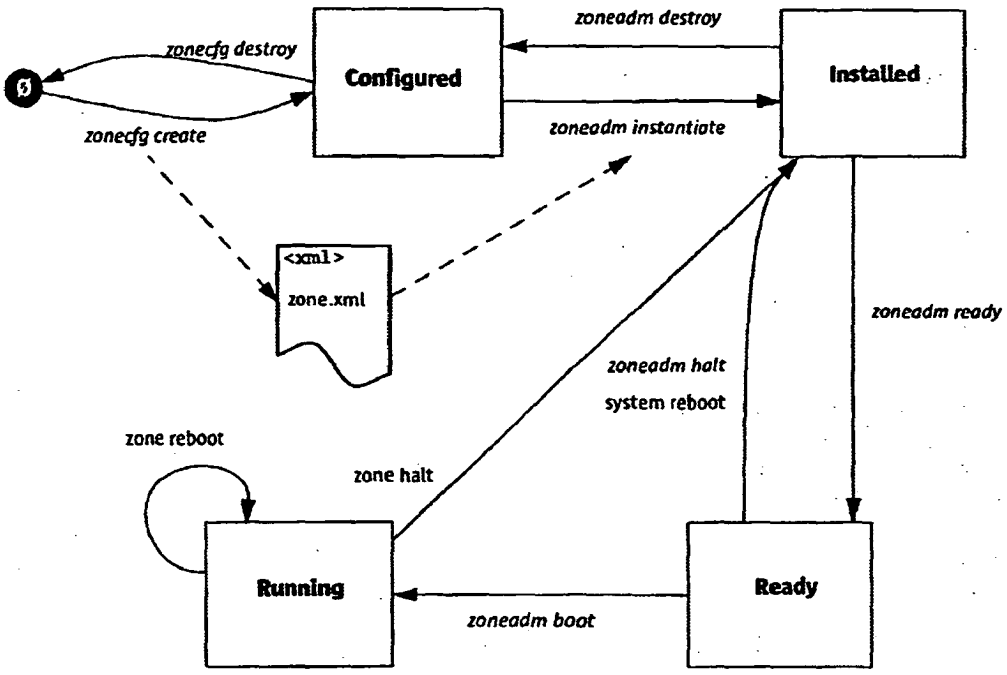

Figure 3.1: Zone State Model

- The READY state: the kernel has created the zone, network interfaces have been plumbed, file systems have been mounted, devices have been configured, but no processes have been started.
- The RUNNING state: processes have been started.

3.2 Zone Configuration

The new `zonecfg(1M)` command is used to create and modify a zone's configuration. `zonecfg` is concerned with general syntax: that the resources and properties specified could make sense on *some* system. Once a configuration has been created, `zoneadm(1M)` will be concerned with specifics: that the resources and properties make sense on that *particular* system.

`zonecfg` operations consist of creating and destroying zone configurations, adding resources to or removing them from a given configuration, setting properties for those re-

sources, plus the usual informational query, verify, commit and revert operations that one would expect with a tool of this nature; see the man page for details. A brief example is given below; subsequent sections provide longer examples indicating how to configure specific resources and properties in various subsystems.

```
# zonecfg -z nebbiolo-z1
zonecfg> import SUNWdefault
zonecfg> add rootpath /aux0/my-zone
zonecfg> add net myhme
zonecfg> setprop net myhme physical hme0
zonecfg> setprop net myhme address 129.146.126.203
zonecfg> verify
zonecfg> commit
zonecfg> ^D
```

3.2.1 Configuration Data

At present, `zonecfg(1M)` uses a project-private XML file to store its configuration data (see Appendix B for the Document Type Definition); later, a Greenline repository or other more general directory-based back-end is envisioned. These data consist of various *resources* some of which have *properties*. The resource types are below; see the man page for the resource type syntax and the details on property types.

zone name Each zone has a unique name; see the next section for details on the allowed syntax for zone names.

zone id Each zone has a unique non-negative integer by which the kernel keeps track of it. Zero is for the global zone; all other numbers less than 100 are reserved.

root path Each zone has a path to its root directory relative to the global zone's root directory. At installation time, this directory will be required to have the same ownership and permissions as the root directory in the global zone: owned by root, mode 755. Unprivileged users in the global zone will be prevented from traversing a non-global zone's file system hierarchy; see Section 7.4 for further discussion of such issues.

file systems Each zone may have various file systems which should be mounted when the zone transitions from the INSTALLED state to the READY state.

network interfaces Each zone may have network interfaces which should be plumbed when the zone transitions from the INSTALLED state to the READY state.

devices Each zone may have devices which should be configured when the zone transitions from the INSTALLED state to the READY state.

resource controls Each zone may have resource controls which should be enabled when the zone transitions from the INSTALLED state to the READY state.

Other resource types may be added; see Section 4.4 for discussion of one such set related to initial zone configuration.

3.2.1.1 Wherefore art thou Verify?

Some resource types have required property types, as detailed in `zonecfg(1M)`. To keep the user interface from being overly complex, a two step process exists: resources are *added*, then properties of those resources are *set*. As such, a configuration of necessity goes through an incomplete state once a resource has been added but its required properties have not yet all been set. To prevent such partial configurations, `zonecfg(1M)` has a `verify` subcommand which reports any incompleteness; an incomplete configuration will not be committed to stable storage.

3.2.2 Zone Names and IDs

Each zone, including the global zone, is assigned a zone name. The rules for a valid zone name are similar to those for host names: they must start with an alpha-numeric character, and all remaining characters must consist of alpha-numerics plus "-" and "_". The name "global" is reserved, as is anything beginning with "SUNW".

Each zone name is mapped to a unique numeric zone identifier (or `zoneid`); this id uniquely identifies the zone on a given system. IDs 0-99 are reserved for use by Sun.

The global zone always has the name "global" and is always mapped to id 0. Zone ids and zone names are under the control of the global administrator, and are generally of little interest within non-global zones. To distinguish between zones, however, the `zonename(1)` command can be useful.

It is worth noting that each zone also has a node name (i.e. returned by `uname -n` in the zone). The node name for each zone is under the control of the zone administrator, and generally of little interest to other zones. The node name is completely independent of the zone name.

3.2.3 Database for Zones

In addition to the name and id stored in each zone's configuration file, a master index mapping these names and ids will be kept in a separate file. See man pages for `getzonebyname(3C)` for details on the interfaces provided to access these data. Prior to the completion of this project, we expect to extend the current interface to include other attributes stored in the configuration file.

3.3 Zone Administration

As explained above in Section 3.1, once a zone has been configured, it is ready to be installed onto the system, then booted, etc. The `zoneadm(1M)` command is used to install, list, boot and halt zones. Below is a series of sample sessions.

3.3.1 Listing Zones

First, the zones on the system are listed by the global administrator:

```
# zoneadm -v info
    ZID ZONENAME        STATE           ROOT
      0 global          running         /
    100 nebbiolo-z1     configured      /aux0/my-zone
```

3.3.2 Installing Zones

Next, the administrator verifies that the zone can be installed safely, and installs the zone, which results in the zone being installed under its root path on the global zone's file system:

```
# zoneadm -z nebbiolo-z1 verify
# zoneadm -z nebbiolo-z1 install
# zoneadm -v info
    ZID ZONENAME        STATE           ROOT
      0 global          running         /
    100 nebbiolo-z1     installed       /aux0/my-zone
```

3.3.3 Booting Zones (external view)

Next, the administrator boots the zone; this results in kernel data structures being set up, file systems being mounted, network interfaces being plumbed, devices being configured and processes being started:

```
# zoneadm -z nebbiolo-z1 boot
# zoneadm -v info
    ZID ZONENAME        STATE           ROOT
      0 global          running         /
    100 nebbiolo-z1     running         /aux0/my-zone
```

Refer to `zoneadm(1M)` for further details.

3.3.4 Booting Zones (internal view)

When `zoneadm boot` is run, `zoneadm` launches `zparent(1M)` which is responsible for creating the zone: setting privileges, calling `zone_create(2)`, registering with `devfsadm(1M)`, initializing the console, and launching a new `init(1M)`-like process called `zinit(1M)`, which will in turn take care of initializing various things within a zone, and iterate through the various start-up scripts to "boot" the zone. `zinit` will support `inittab(4)` in a limited fashion. `init` will be modified to exec `zinit` with the same arguments if called from within a zone.

Run-levels, start-up configuration, etc. are all under investigation, but see Chapter 12 for thoughts on how some of this will work. With regard to Greenline [10]'s service model, zones can be thought of as a two-level services: once a zone has been installed, bringing it to the READY state is one level of service; then bringing it from there to the RUNNING state is another level of service on top of that.

3.3.5 Halting Zones

In the event that an administrator wishes to cause the operating system to remove a zone from its table of available system zones, `zoneadm halt` may be used as demonstrated below. The end result of this operation is that the zone is brought back to the INSTALLED state: all processes are killed, devices are unconfigured, network interfaces are unplumbed, file systems are unmounted, and the kernel data structures are destroyed.

```
# zoneadm -v info
    ZID ZONENAME        STATE           ROOT
      0 global          running         /
    100 nebbiolo-z1     running         /aux0/my-zone
# zoneadm -z nebbiolo-z1 halt
# zoneadm -v info
    ZID ZONENAME        STATE           ROOT
      0 global          running         /
    100 nebbiolo-z1     installed       /aux0/my-zone
```

In the event that the system state associated with the zone cannot be destroyed, the *halt* operation will fail halfway, leaving the zone in an intermediate ZOMBIE state, somewhere between RUNNING and INSTALLED. In this state there are no user processes or kernel threads doing work on behalf of the zone, and none may be created. The administrator is expected to manually intervene in such cases where automatic shutdown fails.

The most common cause of such failure is being unable to unmount all file systems. Traditional Solaris executes a half-hearted "umountall" while shutting down, which does not necessarily unmount all mounted file systems. This is OK since the system state is then destroyed; zones, however, must clean up after themselves such that no mounts performed while booting the zone or during zone operation linger once the zone has been shutdown.

This becomes complicated when dead NFS servers, forcibly unmounted file systems, and the autofs semantics of periodic in-kernel unmounts rather than explicit userland unmounts are considered. Even though `zoneadm` makes sure there are no processes executing in the zone, the unmounting may fail if processes in the global zone have open files in the zone. The global administrator will need to use available tools such as `pfiles(1)` and `fuser(1)` to find the offending process and take appropriate action.

3.3.6 Rebooting Zones

Should a global administrator wish to reboot a given zone for any reason, `zoneadm reboot` may be used as demonstrated below. This will essentially `halt` the zone, then `boot` it anew, going through the same state transitions, with the same results, as described above. The *end* result, however, is as expected, the same as after `boot`: kernel data structures are set up, file systems are mounted, network interfaces are plumbed, devices are configured and processes are started:

```
# zoneadm -v info
     ZID ZONENAME          STATE            ROOT
       0 global            running          /
     100 nebbiolo-z1       running          /aux0/my-zone
# zoneadm -z nebbiolo-z1 reboot
# zoneadm -v info
     ZID ZONENAME          STATE            ROOT
       0 global            running          /
     100 nebbiolo-z1       running          /aux0/my-zone
```

3.3.7 Automatic Zone Booting

In addition to the manual process for booting zones described above, there needs to be a way to specify that certain zones should automatically be booted when the global zone is booted. The specifics of this mechanism have not yet been determined.

3.4 Zone Login

The `zlogin(1)` command can be used by the global administrator to login from the global zone to any other given zone.

3.4.1 Zone Console

Each zone maintains a simple virtual console; as in a normal Solaris instance, standard console I/O (including zone boot messages) are directed to this console. The zone console

may be accessed using the zlogin(1) command with the -C option, as follows; in this case, the zlogin session was started immediately after zoneadm boot was issued; this allows the administrator to see bootup messages from the zone:

```
# zonename
global
# zlogin -C nebbiolo-z1

SunOS Release 5.10 Version k10_29 64-bit
Copyright 1983-2002 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.
starting rpc services: rpcbind done.
syslog service starting.
The system is ready.

nebbiolo-z1 console login: root
Password:
Mar 24 01:44:00 nebbiolo-z1 login: ROOT LOGIN /dev/console
Last login: Mon Mar 24 01:43:27 on console
Sun Microsystems Inc.   SunOS 5.10      k10_29  Mar. 24, 2003

# tty
/dev/console
# zonename
nebbiolo-z1
# ~.
Connection to zone nebbiolo-z1 console closed.

# zonename
global
```

3.4.2 Interactive Mode

While it is useful to do some actions "on console", a more convenient method for accessing the zone and for executing commands inside the zone is provided by the basic zlogin(1) command. In zlogin's *interactive mode*, a new pseudo-terminal will be allocated for use inside the zone. Interactive mode is activated when the user does not include a command to be issued:

```
# zonename
global
# tty
```

```
/dev/pts/3
# zlogin nebbiolo-z1
Last login: Thu Mar 27 00:03:59 on console
Sun Microsystems Inc.   SunOS 5.10      k10_29  Mar. 24, 2003

# zonename
nebbiolo-z1
# tty
/dev/pts/13
#
```

Non-interactive mode is appropriate for shell-scripts which administer the zone, and does not allocate a new pseudo-terminal. Non-interactive mode is enabled when the user supplies a command to be run inside the zone:

```
# zonename
global
# zlogin nebbiolo-z1 zonename
nebbiolo-z1
# zlogin nebbiolo-z1 tty
not a tty
#
```

The behavior of interactive and non-interactive modes is closely consistent with the `ssh(1)` command. Supporting ssh's `-T` (force pseudo-tty allocation) and `-t` (disable pseudo-tty allocation) options is under consideration.

It is occasionally suggested that the project deliver a pair of `zlogin`-style commands, mimicking the behavior of `rlogin(1)` and `rsh(1)`. The project team believes that the model provided by `ssh(1)` is cleaner and more modern; in any case, the obvious command name, `zsh`, is already in wide use.

3.4.3 Zone Console Design Considerations

Section 3.4.1 demonstrates that zones export a virtualized console. More generally, the system's console is an important and widely referenced notion; as seen in the previous section, the zone console is a natural and familiar extension of the system for administrators. In general, the zone or system console has the following properties:

- Applications may open and write data to the console device.
- The console remains accessible when other methods of login (such as `telnet(1)`) fail.
- The system administrator uses the system console to interact with the system when no other system services are running.

- The console captures messages issued at boot-up.
- The console remains available even when the operating system has failed or shut down.
- The console need not have a process consuming data written to it in order to drain its contents.
- Windowing systems make use of the SRIOCSREDIR ioctl and the redirmod STREAMS module to redirect console output to a designated terminal in the window system.

In the current implementation, the console is emulated using some symlinks and the pts(7D) driver; this fails several of the criteria above. We are currently investigating how to build a loopback STREAMS device to better satisfy these requirements.

3.4.4 Zone System Controller

One idea under review is that the global zone should act as a "system controller" for the zones it hosts. This would allow zone administrators a way to establish a connection to the zone console without needing a high level of privilege (or even a regular shell account) on the host system. This would also be valuable for recovery if the zone's network services fail or become misconfigured; for example, a zone administrator who disables inetd(1M) in a particular zone might become locked out of the zone.

We believe that with some further effort, the zlogin(1) command can be used to hand off control from global to non-global zone as a user shell process in /etc/passwd. As a result, the zone console can be made accessible via telnet(1) or ssh(1):

```
# telnet nebbiolo
login: nzl
Password:

nebbiolo-z1 console login: root
Password:
Mar 24 02:20:15 nebbiolo-z1 login: ROOT LOGIN /dev/console
Last login: Mon Mar 24 01:44:00 on console
Sun Microsystems Inc.   SunOS 5.10      k10_29  Mar. 24, 2003

# zonename
nebbiolo-z1
# ~.
Connection to zone nebbiolo-z1 console closed.
Connection to nebbiolo closed by foreign host.
```

We expect to refine this concept in the near future, although this feature is not considered a requirement for this project.

3.4.5 Remote Login

Remote login to a zone depends on the selection of network services established by the zone administrator. By default, logins via `ssh(1)`, `telnet(1)`, `rlogin(1)`, etc. function normally.

3.5 Monitoring and Controlling Zone Processes

Some system utilities are enhanced to provide monitoring and/or filtering of zones:

- `prstat(1M)` is enhanced with `-z` and `-Z` options. `-z [zone ...]` allows filtering by zone name or ID. `-Z` reports information about processes and zones in a split screen view. It is analogous to the `-T` and `-J` options presently available. Figure 3.2 shows the `-Z` option in operation.
- `pgrep(1)` and `pkill(1)` are enhanced with a `-z` option. `-z [zone ...]` allows filtering by zone name or ID.
- `ps(1)` allows selection of processes associated with a given zone with `-o zone` and `-o zoneid`, which print zone names and numeric identifiers, respectively.
- `priocntl(1)`, `renice(1)`, and `poolbind(1M)` have been extended to add a `-i zone` option, which can be used to apply changes to scheduling parameters and resource pool bindings to all processes within a given zone.

```
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
101686 root     4576K 4312K cpu0    54    0   0:00:00 0.7% prstat/1
101689 root      328K  272K sleep   59    0   0:00:00 0.4% sh/1
100811 root     2992K 2360K sleep   59    0   0:00:00 0.1% nscd/18
101687 root     1864K 1360K sleep   59    0   0:00:00 0.0% in.rlogind/1
100778 root     2320K 1328K sleep   59    0   0:00:00 0.0% rpcbind/1
100371 root     2312K 1584K sleep   59    0   0:00:17 0.0% mibiisa/7
100243 root     3000K 2048K sleep   59    0   0:00:00 0.0% nscd/19
101675 root     2536K 2000K sleep   59    0   0:00:00 0.0% inetd/1
100222 root     3632K 1656K sleep   59    0   0:00:00 0.0% syslogd/13
101651 root     2192K 1296K sleep   59    0   0:00:00 0.0% rpcbind/1
100318 root     3672K 1272K sleep   59    0   0:00:00 0.0% htt_server/2
100225 root     4096K 1976K sleep   59    0   0:00:04 0.0% automountd/3
100224 root     2384K 1296K sleep   59    0   0:00:00 0.0% cron/1
100175 root     2136K  608K sleep   59    0   0:00:00 0.0% ypbind/1
100191 root     2632K 1352K sleep   59    0   0:00:00 0.0% inetd/1
ZONEID    NPROC  SIZE   RSS MEMORY      TIME  CPU ZONE
     0       49  113M   55M   5.5%   0:00:25 0.8% global
     1       11   21M   13M   1.3%   0:00:00 0.5% nebbiolo-z1
     2        4 9840K 6080K   0.6%   0:00:00 0.0% nebbiolo-z2

Total: 64 processes, 149 lwps, load averages: 0.13, 0.15, 0.08
```

Figure 3.2: Using prstat to Monitor Zone Activity

Chapter 4

Administration within Zones

In general, administration of the application environment within a zone is identical to current administrative practice. While not every system administrative facility is available, a rich set of configuration options is available to the zone administrator. This chapter highlights some notable administrative facilities and explains how the project supports them.

4.1 Node Name

A system's node name is often used (in conjunction with host naming service) to determine the system's host name. Each zone may have a unique host name under the control of the zone administrator. Hence each zone requires a separate node name under the control of the zone administrator. The node name for each zone is retained by the kernel. The `uname -n` command can be used to report this name, as opposed to the `zonename(1)` command which can be used to report the zone name.

4.2 Name Service Usage within a Zone

Each zone can run any naming service or combination of naming services of its choice, in much the same way that separate systems can do so. This is under the control of the zone administrator. The naming services in different zones are isolated from each other.

NIS and NIS+ make use of a domain name, which is stored within the kernel (`srpc_domain`). This field is changed so that it may be modified per-zone; different zones can have different domain names. The `sysinfo(2)` implementation in the kernel is appropriately changed to save and return domain name based on the caller's zone. No system call is provided to save or return the domain name for zones other than the caller's own zone.

Files used by naming services, such as the `/var/yp` and `/var/nis` reside within a zone's own root file system viewpoint, and hence are isolated between zones.

4.3 Default Locale and Timezone

The default locale and time zone for a zone can be configured independently of those for the global zone (as per `init(1M)`, or the sysidtools described below).

4.4 Initial Zone Configuration

The sysidtools `sysidnet(1M)` and `sysidsys(1M)` are made zone aware, and able to be run in a zone. However, their operation within a zone is limited: they cannot configure IP addresses or routing. `sys-unconfig(1M)` is also available inside a zone, and returns the sysid settings to the unconfigured state; on next zone boot, the sysid tools prompt for the configuration in the normal way.

When a zone is installed, it is left in the unconfigured state, and hence the sysidtools are run the first time a zone is booted. Optionally, it will be possible to specify a jumpstart configuration for a zone, and this will then bypass the sysidtools prompts where the jumpstart configuration contains the relevant answer.

A contract is needed with the Admin/Install consolidation to allow this project to use `sys-unconfig -R` and to create a `/etc/sysidcfg` file. This will be created for project commitment.

4.5 System Log Daemon

The system log is maintained by the daemon `syslogd(1M)`, which reads messages from the `/dev/log` device and sends them to log files or users as specified by the contents of `syslog.conf(4)`. Log messages can be originated by user processes calling `syslog(3C)`, or by kernel subsystems calling `strlog(9F)` or `cmn_err(9F)`. The `syslog(3C)` interface, in turn, writes messages to the `/dev/conslog` device.

Applications within a zone should be able to generate log messages whose disposition is under the control of the zone administrator, so that these messages can be visible to users (particularly administrators) within the zone. This requires modifying the log driver to virtualize `/dev/log`, so that the syslogd daemon running in each zone sees only the messages intended for that zone. The result is that messages generated by applications within a zone are seen by `/dev/log` clients (e.g., syslogd) within the same zone.

Kernel messages are often associated with physical hardware and other aspects of a system that are not relevant within a zone; hence, most kernel messages will be sent to the global zone. For the exceptions, where a message is closely related to the activities of a specific zone, a kernel interface analogous to `cmn_err(9F)` will be created that allows the caller to specify which zone should be the recipient of the message.

An advantage of this approach (which requires changes only to the log driver, not `syslog(3C)` or `syslogd(1M)`) is that third-party versions of `syslogd`, such as `syslog-ng`, will work inside a zone without modification.

4.6 Commands

The process oriented commands (`ps(1)`, `pgrep(1)`, ...) work in a zone, showing only the processes within that zone due to the filtering provided by procfs. Note that `sched` and `init` are special cases which appear to be running in all zones, though there is really only a single instance of each on the system.

`coreadm(1M)` does not have any new options, but its behavior is changed such that when run from a zone (including the global zone), it only affects the zone in which it is run. This is for consistency with the global zone visibility policy described in Section 6.3.

`acctadm(1M)`'s behavior is changed as described in Section 10.1.1.2. Note that the extended accounting subsystem (run in the global zone) collects information for the entire system, including non-global zones.

`fuser(1)` (and the underlying system) is modified to only report processes running in the current zone. If the calling process is running in the global zone with the appropriate flag set it will see information for processes running in all zones.

6046958.050903

Sun Proprietary/Confidential: Internal Use Only

Chapter 5

Security

One of the basic tenets of the zone design is that no process running within a (non-global) zone, even one with super-user credentials (running with an effective user id of 0), is allowed to view or affect activity in other zones. This implies that any operation initiated from within a zone must have an effect that is local to that zone. For example, the following activities will not be allowed within a non-global zone:

- Loading custom kernel modules (those not installed in the system's module search path).
- Rebooting or shutting down the system as a whole.
- Accessing kernel memory through `/dev/kmem`.
- Accessing physical devices (other than those that may be assigned to the zone for its exclusive use).
- Configuring physical network interfaces or network infrastructure (e.g., routing tables).

The security model requires that only a subset of the operations normally restricted to super-user will be allowed within a zone, since many of those operations have a global impact. Operations that will not be allowed include halting or rebooting the system, creating device nodes, and controlling allocation of global system resources. Processes running in the global zone will still have the full set of privileges, allowing them to affect activity in any zone of the system. Effectively, the zone in which a process is running becomes part of its credential information, restricting its capabilities beyond those of processes with identical effective user and group ids running in other zones.

5.1 Credential Handling

This project proposes extending the `cred_t` structure in the kernel, adding a `cr_zone` field indicating the zone id associated with the credential. This field is set for any process en-

tering a zone, and is inherited by the credentials used by all descendant processes within the zone. The `cr_zone` field is examined during privilege checks using credential information, such as `priv_policy(9F)`, `drv_priv(9F)`, and `hasprocperm`. The kernel interface `crgetzoneid(9F)` is used to access the zone id without directly dereferencing the `cred_t` structure.

5.2 Fine-Grained Privileges

The Least Privilege project (PSARC/2002/188) [7] added a facility for breaking up the privileges that the kernel previously granted to processes with an effective uid of 0. The notion of "super-user privilege" is replaced by a *set* of specific privileges, such as the privilege to perform a mount or manipulate processor sets. This means that processes can have the ability to perform certain privileged operations, but not others.

5.2.1 Safe Privileges

The privilege framework allows the restrictions on activity within a non-global zone to be expressed as a subset of privileges that are considered "safe" from a zone standpoint—that is, those privileges that do not allow the exercising process to violate the security restrictions on a zone. Table 5.1 shows this list, and Table 5.2 shows the list of unsafe privileges that will normally only be available within the global zone. Note that the brief descriptions of privileges here are meant to be illustrative, not comprehensive; see the `privileges(5)` man page or PSARC/2002/188 [7] for the full descriptions.

Finally, a few privileges are problematic in some way, generally because a single privilege enables both operations that are safe in a non-global zone, as well as those that are not. We're currently determining how to handle this final set. The details of the problems vary:

`PRIV_NET_RAWACCESS` Needed to implement `ping(1M)` in a zone, but also allows fabrication of IP packets.

`PRIV_SYS_CONFIG` Needed to set coreadm settings, hostname, and domain name, but also for a wide variety of operations related to the hardware platform.

`PRIV_SYS_NFS` Used for both client-related and server-related NFS operations. This isn't a problem unless we are not supporting NFS service inside a non-global zone.

`PRIV_SYS_RESOURCE` Needed for configuring privileged resource controls, but privileged per-zone resource controls should not be able to be modified from within a non-global zone.

We will have a proposal for how to address these issues at commitment.

PRIV_FILE_CHOWN	Allows process to change file ownership.
PRIV_FILE_CHOWN_SELF	Allows process to give away files it owns.
PRIV_FILE_DAC_EXECUTE	Allows process to override execute permissions.
PRIV_FILE_DAC_READ	Allows process to override read permissions.
PRIV_FILE_DAC_SEARCH	Allows process to override directory search permissions.
PRIV_FILE_DAC_WRITE	Allows process to override write permissions.
PRIV_FILE_LINK_ANY	Allows process to create hard links to files owned by someone else [basic].
PRIV_FILE_OWNER	Allows non-owning process to modify file in various ways.
PRIV_FILE_SETDAC	Allows non-owning process to modify permissions.
PRIV_FILE_SETID	Allows process to set setuid/setgid bits.
PRIV_IPC_DAC_READ	Allows process to override read permissions for System V IPC.
PRIV_IPC_DAC_WRITE	Allows process to override write permissions for System V IPC.
PRIV_IPC_OWNER	Allows process to control System V IPC objects.
PRIV_NET_PRIVADDR	Allows process to bind to privileged port.
PRIV_PROC_AUDIT	Allows process to generate audit records.
PRIV_PROC_CHROOT	Allows process to change root directory.
PRIV_PROC_EXEC	Allows process to exec [basic].
PRIV_PROC_FORK	Allows process to fork [basic].
PRIV_PROC_OWNER	Allows process to control/signal other processes with different effective uids.
PRIV_PROC_SESSION	Allows process to send signals outside of session [basic].
PRIV_PROC_SETID	Allows process to set its uids.
PRIV_PROC_TASKID	Allows process to enter a new task.
PRIV_SYS_ACCT	Allows process to configure accounting.
PRIV_SYS_MOUNT	Allows process to mount and unmount file systems.

Table 5.1: Safe Privileges

PRIV_PROC_CLOCK_HIGHRES	Allows process to create high resolution timers.
PRIV_PROC_LOCK_MEMORY	Allows process to lock pages in physical memory.
PRIV_PROC_PRIOCNTL	Allows process to change scheduling priority or class.
PRIV_SYS_AUDIT	Allows process to manage auditing.
PRIV_SYS_DEVICES	Allows process to create device nodes.
PRIV_SYS_IPC_CONFIG	Allows process to increase size of System V IPC message queue buffer.
PRIV_SYS_LINKDIR	Allows process to create hard links to directories.
PRIV_SYS_NET_CONFIG	Allows process to configure network interfaces.
PRIV_SYS_RES_CONFIG	Allows process to configure system resources.
PRIV_SYS_SUSER_COMPAT	Allows process to successfully call third-party kernel modules that use suser().
PRIV_SYS_TIME	Allows process to set system time.

Table 5.2: Unsafe Privileges

5.2.2 Zone Privilege Limits

In order to enable the restriction of privileges within a zone, the `zone_create(2)` system call includes an argument to specify the zone's privilege limit. This is the set of privileges that will be used as a mask for all processes entering the zone, including the process that initiates booting the zone.

Note that the limit on privileges available within a zone does not eliminate the need for restrictions on the objects a zone can access. Privileges can be used to determine whether or not a process can perform a given operation, but if the operation is allowed they do not restrict the objects to which that operation can be applied. (There is a special case involving objects with an effective user id of 0, as described below, but the general rule holds.) For example, the `PRIV_PROC_MOUNT` privilege allows a process to mount file systems; if the process has that privilege, it can mount file systems anywhere in the file system namespace. Zones, on the other hand, primarily restrict the namespace and objects to which operations (even unprivileged operations) can be applied; it is only when such restrictions are not possible that the privileges available within a zone must be limited. In short, privileges and zones are complementary technologies.

5.2.3 Privilege Escalation

The problem of privilege escalation represents an additional complication. In order to prevent a process with a subset of privileges from being able to use those privileges to acquire additional privileges, a number of operations involving root-owned system objects require that the calling process have *all* privileges. For example, write access to root-owned files by a non-root process requires all privileges, as does establishing control over a process with an effective uid of 0. The intent is to prevent non-root processes with some but not all privileges

from using the special treatment of root to escalate privileges; e.g., if a non-root process with the PRIV_FILE_DAC_WRITE privilege is allowed to modify the text of kernel modules, it can cause a module to be loaded that awards it all privileges.

Since no process in a non-global zone will have all privileges, the requirement of all privileges for operations involving root-owned objects presents a problem. On the other hand, since even root within a zone has a restricted set of privileges, no privilege escalation is possible beyond the set of zone's privilege limit; thus, it seems appropriate to change the restriction to be the privilege limit for the zone (or all privileges in the global zone).

Note that certain other operations (e.g., loading kernel modules) require all privileges because they can be used to control the entire system. These operations should continue to require all privileges, regardless of the zone in which the process is running.

5.2.4 Configurability of Zone Privilege Limits

The introduction of privilege sets represents an opportunity for making the privileges available within a zone more dynamic. Now that we have a way of controlling the privileges available within a zone as a set, we can give administrators the ability to modify this set. This might be particularly useful for systems in a trusted data center environment, where administrators may wish to allow certain operations within a zone that would normally be considered unsafe. On the other hand, it represents quite a bit of "rope" for administrators, and may result in unforeseen consequences. This is particularly true if the zone implementation itself depends on the existence and absence of certain privileges; changes to those privileges may result in failure of the internal zone mechanism itself, not just changes to the behavior of applications running within the zone.

In addition, the likelihood that both privileges and zones will evolve in the future suggests a potential problem if privilege sets become part of the zone configuration. Let's say an administrator creates a zone with the default (safe) privileges, plus PRIV_PROC_PRIOCNTL. Then, in a subsequent release of Solaris, a new "safe" privilege is added. When the previous configuration is updated, the update script must attempt to determine whether to include the new privilege in the zone's set, or to leave the configuration as specified. Without some explicit statement of intent (e.g., including a **default** token in `zonecfg(1M)` to to specify the default privileges), it will be difficult to determine the correct behavior.

Although the long-term goal is certainly to provide this flexibility to administrators, we'd like to be sure we've understood all of the possible ramifications prior to doing this. We expect to provide this functionality as a follow-on project once the issues have been fully understood and addressed.

5.2.5 Privilege Requirements for Administering Zones

In order to create, destroy, or enter a zone, the PRIV_SYS_CONFIG privilege is required.

5.3 Role-Based Access Control

Like privileges, the role-based access control (RBAC) facility provides a way of making the super-user model more granular. With RBAC, an administrator can give particular users (or "roles", identities that can be assumed by existing users via `su(1M)` but cannot be used for external login) the right to perform operations that would otherwise require super-user privileges. These operations are expressed either as *authorizations*, rights explicitly checked by applications (usually running setuid root), or using *profiles*, lists of commands that can be executed using `pfexec(1)` or the "profile shells" (`pfsh(1)`, `pfcsh(1)`, `pfksh(1)`). Authorizations differ from privileges in that authorizations are enforced by user-level applications (including the profile shells), rather than the kernel. Like privileges, though, they provide fine-grained control over the set of operations a given process can perform, but not over the objects that those operations can affect. As with privileges, this control complements the restrictions imposed by zones.

The zone administration commands requiring privilege (`zoneadm(1M)`, `zonecfg(1M)`, and `zlogin(1)`) will be placed in a new "Zone Management" rights profile.

5.4 Chroot Interactions

The functionality made available by `chroot(2)` is similar in some ways to zones, in that both provide ways to restrict the part of the file system hierarchy a process and its descendants can access. Chroot, however, has a number of problems from a security perspective. In particular, any process that is given super-user privileges can easily escape a chroot restriction. The problem is that chroot calls don't "nest" safely. A process inside a chrooted environment can call chroot to change its working directory to something "below" the current working directory, then make successive `chdir("..")` calls until it reaches the real root directory for the system. This works because the check to determine whether a process is escaping its chroot restriction works by processing a pathname comparing the directory being traversed in each component with the root directory that has been set for the process; if the process has access to a directory "above" its root directory, the check will be bypassed.[1]

This issue is addressed for zones in two ways. One is that a process inside a zone (other than the global zone) cannot enter another zone; this prevents a process running as super-user in a zone from escaping the zone's root directory restriction in a manner similar to what is possible with chroot. The other is that the zone's root directory (represented by the `zone_rootvp` field in the kernel's `zone_t` structure) is distinct from the root directory set by chroot for processes within a zone (represented by the `u_rdir` field in the kernel's `user_t` structure). Both restrictions are checked when traversing pathname components; this means that chroot can be used within a zone, but a process that escapes from its chroot restriction

[1] Even if we were to somehow fix this problem, a process with super-user privileges inside a chroot restriction could still escape by using `mknod(2)` to create a `/dev/kmem` device node and writing to the appropriate kernel data structure.

will still be unable to escape the zone restriction.

5.5 Audit

The Solaris audit facility provides the ability to create a log of security relevant operations performed by processes on an appropriately configured Solaris instance.

We are still determining the specifics of how audit should work with respect to zones, and whether those changes will be included in this case or be part of a separate case. The following are some initial thoughts regarding how they might integrate.

Zones will be able to configure their own individual audit logs. If any zone wishes to use the Solaris audit facility, audit must be enabled and partially configured in the global zone when the Solaris instance is booted. Audit configuration is split into two parts: the global configuration parameters that control the audit subsystem as a whole, and the per-zone configuration parameters. The global configuration parameters include the following `auditconfig(1M)` commands, as well as the corresponding `auditon(2)` commands:

-conf (`auditon A_SETCLASS`) configures the event to class mapping.

-aconf (`auditon A_SETKMASK`) configures the kernel non-attributable events.

-setcond (`auditon A_SETCOND`) turns on/off auditing.

Other parts of the Solaris audit facility are configured within each zone (including the global zone) that wishes to record audit logs. The per-zone configuration includes administering `audit_control(4)`, `audit_user(4)`, `audit_warn(1M)`, and `audit_startup(1M)` within each zone, as well as use of the `auditconfig(1M)` commands not listed above.

Each zone that desires audit trail recording will run its own audit daemon, `auditd(1M)`, and has its own audit trail files/directories (`audit.log(4)`). Even if non-global zones run an audit daemon, there is no requirement that the global zone run an audit daemon, just that the Solaris instance load the audit module and the global zone configure the event to class mapping and non-attributable events.

A new audit token, `AUT_ZONEID` (0x16), will be added to audit records to indicate which zone produced them.

All Solaris audit interfaces (`audit(2)`, `getaudit(2)`, `setaudit(2)`, `getaudit_attr(2)`, `setaudit_addr(2)`, `getauid(2)`, `setauid(2)`, `auditsvc(2)`), except for `auditon(2)` subcommands noted above, operate in all zones. Note: `auditon(2)` and `auditctl(2)` are historic synonyms. `auditctl(2)` is not documented.

Chapter 6

Process Model

As described in Chapter 5, processes within one zone (other than the global zone) must not be able to affect the activity of processes running within another zone. This also extends to visibility; processes within one (non-global) zone should not even be able to see processes outside that zone.[1] This can be enforced by restricting the process id space exposed through `/proc` accesses and process-specific system calls (`kill(2)`, `priocntl(2)`, etc.). If the calling process is running within a non-global zone, it will only be able to see or affect processes running within the same zone; applying the operations to any other process ids will return an error.

Note that the intent here is not to try to prevent all possible covert channels for passing information between zones. Given the deterministic algorithm for assigning process id's, it would be possible to transmit information between two zones on an otherwise idle system by forking processes periodically and monitoring the assigned process ids. The intent is to prevent unintentional information flow from one zone to another, not to block intentionally constructed (from both sides) low-bandwidth channels of information.

6.1 Signals and Process Control

As mentioned above, processes in one zone will not be able to affect the activity of those in other zones (with the exception that processes in the global zone can affect the activity of other processes). This is the case even if the acting process has an effective user id of 0 or is executing within an RBAC profile. As a result, attempts to signal or control (via `/proc` or other mechanisms) processes in other zones will fail. Such attempts will fail with an error code of ESRCH (or ENOENT for `/proc` accesses), rather than EPERM; this avoids revealing the fact that the selected process id exists in another zone. More importantly, it ensures that an application running in a zone sees a consistent view of system objects; there aren't objects that are visible through some means (e.g., when probing the process ID space using `kill(2)`) but not others (e.g., `/proc`).

[1] With the exception of `sched` and `init`, as noted in Section 6.3.

6.2 Global Zone Visibility and Access

The dual role of the global zone, acting as both the default zone and as a zone for administrative control, can cause certain problems. For some applications running in the global zone, the scope of the application should be limited to system objects associated with the global zone. For example, a shutdown script that uses `pkill(1)` to signal a service of a given name should be limited to acting on processes within the same zone, even within the global zone. This also ensures that the visibility of kernel-based objects like processes matches that of user-level based objects like users (e.g., the data stored in `/var/adm/utmpx`).

For other applications, a system-wide scope is more desirable. For example, an administrator wishing to monitor the system-wide resource usage might wish to look at process statistics for the whole system. A view of just global zone activity will miss relevant information from other zones in the system that may be sharing some or all of the system resources. Such a view is particularly important when the use of relevant system resources (CPU, memory, swap, I/O) is not strictly partitioned using resource management facilities.

The chosen solution to this problem is to allow processes in the global zone to choose their preferred scope and visibility. By default, access to system objects from the global zone will be restricted to those objects associated with the global zone. For example, a process (regardless of privilege) examining `/proc` within the global zone will see only processes within the global zone. Similarly, `kill(2)` will be restricted to acting only on processes in the global zone. Likewise, a process examining `/etc/mnttab` will see only file system mounts associated with the global zone, a process calling `msgids(2)` or equivalent will see only System V IPC identifiers associated with the global zone, and a process performing a `SIOGLIFCONF ioctl(2)` to IP will only see network interfaces associated with the global zone.

By use of a per-process flag, however, an application or administrator can select system-wide visibility. The new flag, `PROC_ZONE_AWARE`, has been added to the `setpflags(2)` interface introduced by PSARC/2002/188. When this flag has been set on a process in the global zone, the process will have a global view of the virtualized system objects: processes, file system mounts, System V IPC objects, and network interfaces. That is, the operations involving those objects will have global visibility; `/proc` will show all processes, `kill(2)` can be applied to any process (subject to permission restrictions), `msgids(2)` will return all message queue identifiers, etc.. The flag is off by default, and is inherited across `fork(2)` and `exec(2)`. It will have no effect on processes running in non-global zones.

We are also planning to add a command (tentatively named `pzone`) that can be used to launch processes with this flag set. This could be used to provide global visibility to third-party and other tools that have not been modified to call `setpflags(2)`; for example, a process monitoring tool such as `top` could be able to view all processes in the system. The launcher will simply set the `PROC_ZONE_AWARE` flag and exec the indicated command; details will be made available for the commitment review.

Since any process can set the `PROC_ZONE_AWARE` flag, unprivileged processes in the global zone will be able to see processes in different zones with the same user id. This raises a

potential security issue—since other zones can have different name service configurations, processes with the same user id in different zones may actually belong to different users (much as the same user id can be used for different users on separate systems). Hence, it is advisable to prevent unprivileged processes in the global zone from exerting control over processes in other zones. The solution is to treat processes in different zones as if they had different effective user ids, even if the user ids match; attempts by a process in the global zone to signal or control a process in a non-global zone will require the `PRIV_PROC_OWNER` privilege, or all privileges if the indicated process has an effective user id of 0.

6.3 /proc

The `/proc` file system (or procfs) will be enhanced to provide the process visibility and access restrictions mentioned above and information about the zone association of processes. The process access restrictions are based on a mount option, `-o zone=<zoneid>`, that specifies that the instance of procfs being mounted will only contain processes associated with the specified zone. The mount point for that instance will generally be the "proc" subdirectory of the corresponding zone root directory; this allows processes running within the zone to access `/proc` just as they would previously, except that they only see processes running within the same zone. If the `/proc` mount is issued from inside a non-global zone, the `-o zone=<zoneid>` option is implicit.

If the `/proc` file system is mounted from within the global zone, and no `-o zone` option is specified, then the file system will operate in one of two modes. If the process accessing the file system has the `PROC_ZONE_AWARE` flag set, then the file system will contain all processes in the system; otherwise, it will only contain processes in the zone of the process performing the access.

The procfs entries are further "doctored" to prevent leakage of process information from the global zone and to provide a consistent process tree within the zone. In particular, processes 0 (`sched`) and 1 (`init`) are visible within every zone; any process whose parent does not belong to the zone appears to be parented by process 1. This allows tools like `ptree(1)` that expect a tree of processes (rather than a forest) to continue to work without modification. (We could fix `ptree`, but assume that other applications have similar expectations.) It also prevents exposure of the real parent process ids (belonging to the global zone) within the zone.

Another approach to limiting access to certain processes within an instance of procfs would be to do the filtering based on the zone context of the opening process, rather than through use of a mount option. That would mean that, when a process in the global zone opened a procfs instance associated with another zone, it would actually see all processes in the system rather than just the ones associated with that zone. This was thought to be more confusing than the mount option approach, where a given procfs instance will export the same processes regardless of the context of the reader.

The files exported by procfs will also be enhanced to include data about the zone with

which each process is associated. In particular, a zone id will be added to the pstatus and psinfo structures (available by reading the corresponding files in procfs). The zone id replaces a pad field in each structure, so it will not affect binary compatibility. This addition allows processes in the global zone to determine the zone associations of processes they are observing or controlling.

```
# zonename
global
# zoneadm -v info
     ZID ZONENAME         STATE            ROOT
       0 global           running          /
     100 nebbiolo-z1      running          /aux0/my-zone
# ps -e -o pid,zoneid,comm
   PID ZONEID COMMAND
     0      0 sched
     1      0 /etc/init
     ...
100180      0 /usr/lib/netsvc/yp/ypbind
100228      0 /usr/lib/autofs/automountd
100248      0 /usr/sbin/nscd
# pzone ps -e -o pid,zoneid,comm
     0      0 sched
     1      0 /etc/init
     ...
100180      0 /usr/lib/netsvc/yp/ypbind
100228      0 /usr/lib/autofs/automountd
100248      0 /usr/sbin/nscd
103152    100 /usr/sbin/inetd
     ...
103148    100 /usr/lib/autofs/automountd
103141    100 /usr/lib/netsvc/yp/ypbind
# zlogin nebbiolo-z1 ps -e -o pid,zoneid,comm
   PID ZONEID COMMAND
     0      0 sched
     1      0 /etc/init
103148    100 /usr/lib/autofs/automountd
103141    100 /usr/lib/netsvc/yp/ypbind
103152    100 /usr/sbin/inetd
103139    100 /usr/sbin/rpcbind
103143    100 /usr/sbin/nscd
```

Figure 6.1: /proc viewed from the global zone and a non-global zone

Chapter 7

File Systems

Virtualization of storage in a zone is achieved via a restricted root, similar to the `chroot(2)` environment at the file system level. Processes running within a zone will be limited to files and file systems that can be accessed from the restricted root. Unlike chroot, a zone will not be escapable; once a process enters a zone, it and all of its children will be restricted to that zone and associated root.

The loopback file system (lofs) provides a useful tool for constructing a file system namespace for a zone. This can be used to mount segments of a file system in multiple places within the namespace; for example, `/usr` could also be mounted underneath a zone root. The use of lofs in installing a zone is further described in Chapter 12.

7.1 Configuration

Generally speaking, the set of file systems mounted in a zone is the set of the file systems mounted by the global zone into the zone (such mounts are generally done while setting up the zone) plus the set of file systems mounted from within the zone (for instance, the file systems specified in a zone's `/etc/vfstab`, as well as autofs and autofs-triggered mounts and mounts explicitly performed by zone administrator). Certain restrictions are placed on mounts performed from within a non-global zone to prevent the zone administrator from denying service to the rest of the system, or otherwise negatively impacting other zones.

7.1.1 Zonecfg File System Configuration

The global administrator can specify a number of mounts to be performed when the zone is transitioning from the INSTALLED state to the READY state. The interface for specifying that `/dev/dsk/c0t0d0s7` in the global zone is to be mounted as `/var/tmp` in zone *nebbiolo-z1*, and that the file system type to use should be UFS, mounted with logging enabled is:

```
zonecfg> add fs /var/tmp
zonecfg> setprop fs /var/tmp special /dev/dsk/c0t0d0s7
```

```
zonecfg> setprop fs /var/tmp type ufs
zonecfg> setprop fs /var/tmp options logging
zonecfg> info fs /var/tmp
fs: /var/tmp
        special: /dev/dsk/c0t0d0s7
        type: ufs
        options: logging
```

7.2 Size Restrictions

This project does not attempt to provide limits on how much disk space can be consumed by a zone, through zone-wide quotas or otherwise. The global administrator is responsible for space restriction. Administrators interested in this functionality have a number of options, including:

lofi A global administrator may place the zone on a `lofi(7D)`-mounted partition, limiting the amount of space consumable by the zone to that of the file used by lofi.

Soft Partitions allow disk slices or logical volumes to be divided into up to 8192 partitions. A global administrator may use these partitions as zone roots, and thus limit per-zone disk consumption.

ZFS [4] plans to permit the creation of a virtually unlimited number of file systems from a storage pool, and will also be an option for global administrators.

7.3 File System-Specific Issues

There are certain security restrictions on mounting certain file systems from within a zone, while others exhibit special behavior when mounted in a zone. The list of modified file systems, the issues surrounding them, and how we plan to deal with these issues are summarized below.

autofs Each zone runs its own copy of automountd, with the auto maps and timeouts under the zone administrator's control. Since the zone's file system namespace is really only a subset of that of the global zone, the global zone could possibly create auto maps that reference the non-global zone, or traverse into non-global zones and attempt to trigger mounts. This would cause a number of complications, which we circumvent by saying that triggering a mount in another zone (i.e., crossing an autofs mount point for a non-global zone from the global zone) is disallowed, and that the lookup request fails. Note that such situations cannot arise without the participation of the global zone.

Certain autofs mounts are created in the kernel when another mount is triggered. For example, the autofs mount `/net/rankine/export1` is created in the kernel when the NFS mount `/net/rankine` is triggered. Such mounts can not be removed via the regular `umount(2)` interface since they must be mounted or unmounted as a group to preserve semantics. There is a kernel thread that periodically wakes up and attempts to communicate with automountd in order to remove such mounts, but no interface to explicitly attempt to remove such mount points. We have created a private interface to provide this functionality that is needed for zone shutdown.

mntfs As shown in Figure 7.1, mntfs is modified such that the set of file systems visible via `mnttab(4)` is the set of file systems mounted in the zone, plus an entry for "/". Mount points with a "special device" (i.e., `/dev/rdsk/c0t0d0s0`) not accessible from within the zone have their special device set to the same as the mount point. mntfs takes a *zoneid* argument similar to what is described for procfs in Section 6.3.

Certain pathnames used as mount points or special devices can cause the *mntent family of libc functions to get confused and fail. Most notable are pathnames that include white space, which cause the userland parser to get confused. While this is not much of a problem on traditional Solaris systems where only the superuser may perform mounts, a malicious zone could cause applications in the global zone with global visibility to fail by "corrupting" mnttab. Since applications by default do not have global visibility, the scope of this problem is somewhat alleviated, although not eliminated. This problem can probably be best solved through eliminating the string parsing of `/etc/mnttab` and by creating a set of private ioctls to be used directly by `getmntent(3C)` and friends.

When mounted from within a zone, mntfs file systems behave as though mounted with *"-o zoneid=*myzoneid*"*.

NFS Due to an existing limitation in NFS documented in the `mount_nfs(1M)` man page, an NFS server should not attempt to mount its own file systems, hence it will not be possible for a zone to NFS mount a file system exported by the global zone. An attempt to do so will result in an error.

In addition, NFS mounts from within a zone behave (implicitly) as though mounted with the *nodevices* option described in Section 9.6.4.

As yet, there is no requirement for a zone to be an NFS server, though this could arise in the future. If such functionality were delivered it would involve virtualizing the NFS and related subsystems such that each zone could act as an NFS server.

procfs See Section 6.3 for a full description of `/proc` modifications.

When mounted from within a zone, procfs file systems behave as though mounted with *"-o zoneid=*myzoneid*"*.

tmpfs Although a "virtual" file system, a malicious zone administrator could use tmpfs to consume all available swap on the system. Permitting tmpfs mounts from within a zone is contingent on devising a mechanism to limit the total swap consumable by tmpfs mounts in a zone.

In addition to the denial-of-service possible by consuming all of swap, a zone may consume a lot of physical memory on the machine by exploiting the fact that inodes on a tmpfs file system are always kept in core, and creating very many small files. This is actually a problem in stock Solaris as well, although there are certain threshold values in the kernel that prevent tmpfs from using all of physical memory. In the absence of explicit per-zone limits, one zone would be able to cause tmpfs file creations in another zone to fail. See Section 10.1.2 for a discussion about per-zone resource limits.

lofs Read-only lofs mounts traditionally did not prevent read-write access to files, a problem that has recently been rectified in PSARC/2001/718 [22], allowing the project to take advantage of lofs during zone installation (see Chapter 12).

Note that the scope of what can be mounted via lofs is limited to the portion of the file system visible to the zone. Hence, there are no restrictions on lofs mounts in a zone.

UFS, hsfs, pcfs These are file system types which (due to bad metadata or other problems with the backing physical device) may cause the system as a whole to fail, and hence can not safely mounted from within a zone. They may, however be mounted in a zone if an appropriate block device is exported to the zone. Hence, for such file systems to exist within a zone they must either be mounted directly by or with the explicit consent (expressed in the zone configuration profile) of the global zone administrator.

7.4 File System Traversal Issues

As mentioned earlier, a zone's file system namespace is a subset of that accessible from the global zone. Global zone processes accessing a zone's file system namespace can open up a host of problems on the system.

In general we discourage processes in the global zone from traversing a non-global zone's file system hierarchy: by insisting on the zone root's parent directory being owned, readable, writable, and executable by root, and restricting access to directories exported by `/proc` 6.3.

The following are highlighted as potential issues avoided by restricted access into the zone's file system namespace, but that should be taken into account by the global administrator.

7.4.1 Security

As zone administrators can set the *setuid* bit on executables, an unprivileged process in the global zone could coordinate with a privileged process in a non-global zone, effectively giving

the process in the global zone all privileges.

7.4.2 Zone shutdown

Cross-zone file accesses may cause certain complications during zone shutdown. As described in Section 3.3.5, all file systems mounted in the zone's namespace must unmounted before the zone can be fully shutdown. While certain file systems (such as NFS) support forcible unmounts, many do not. File systems with files open due to access from the global zone will not be able to be unmounted, hence cross-zone access can interfere with zone rebooting or shutting down.

7.4.3 Autofs

As noted earlier in Section 7.3, attempting to access autofs nodes mounted for another zone will fail. The global administrator should thus take care to not have auto maps that descend into other zones.

```
# zonename
global

# zoneadm -v info
     ZID ZONENAME         STATE            ROOT
       0 global           running          /
     100 nebbiolo-zi      running          /aux0/my-zone

# cat /etc/mnttab
/dev/dsk/c0t0d0s0       /       ufs rw,intr,largefiles,logging,xattr,onerror=panic,suid,dev=800000 1028243575
/devices        /devices        devfs   dev=9cbc0000    1028243566
/proc   /proc   proc    dev=9cc00000    1028243572
mnttab  /etc/mnttab     mntfs   dev=9ccc0000    1028243572
fd      /dev/fd fd      rw,suid,dev=9cd00001    1028243575
swap    /var/run        tmpfs   xattr,dev=1     1028243596
swap    /tmp    tmpfs   xattr,dev=2     1028243598
jurassic.eng:/export/home14/ozgur       /home/ozgur     nfs intr,nosuid,noquota,xattr,dev=9cec0043  1028939560

# cat /aux0/my-zone/etc/mnttab
/       /       ufs rw,intr,largefiles,logging,xattr,onerror=panic,suid,dev=800000 1028243575
/usr    /usr    lofs    rw,suid,dev=800000      1028243598
proc    /proc   proc    zoneid=100,dev=9cc00000      1028570870
fd      /dev/fd fd      rw,suid,dev=9cd00004    1028570870
/opt    /opt    lofs    rw,suid,dev=800000      1028570870
/sbin   /sbin   lofs    rw,suid,dev=800000      1028570870
swap    /tmp    tmpfs   xattr,dev=7     1028570870
swap    /var/run        tmpfs   xattr,dev=8     1028570870
mnttab  /etc/mnttab     mntfs   zoneid=100,dev=9ccc0000      1028570870
taxman.eng:/web /net/taxman.eng/web     nfs intr,nosuid,grpid,xattr,dev=9cec0020    1028572145

# pzone cat /etc/mnttab
/dev/dsk/c0t0d0s0       /       ufs rw,intr,largefiles,logging,xattr,onerror=panic,suid,dev=800000 1028243575
/devices        /devices        devfs   dev=9cbc0000    1028243566
/proc   /proc   proc    dev=9cc00000    1028243572
mnttab  /etc/mnttab     mntfs   dev=9ccc0000    1028243572
fd      /dev/fd fd      rw,suid,dev=9cd00001    1028243575
swap    /var/run        tmpfs   xattr,dev=1     1028243596
swap    /tmp    tmpfs   xattr,dev=2     1028243598
proc    /aux0/my-zone/proc      proc    zoneid=100,dev=9cc00000 1028570870
fd      /aux0/my-zone/dev/fd    fd      rw,suid,dev=9cd00004 1028570870
/opt    /aux0/my-zone/opt       lofs    rw,suid,dev=800000 1028570870
/sbin   /aux0/my-zone/sbin      lofs    rw,suid,dev=800000 1028570870
swap    /aux0/my-zone/tmp       tmpfs   xattr,dev=7     1028570870
swap    /aux0/my-zone/var/run   tmpfs   xattr,dev=8     1028570870
mnttab  /aux0/my-zone/etc/mnttab mntfs  zoneid=100,dev=9ccc0000 1028570870
taxman.eng:/web /aux0/my-zone/net/taxman.eng/web nfs intr,nosuid,grpid,xattr,dev=9cec0020    1028572145
jurassic.eng:/export/home14/ozgur       /home/ozgur     nfs intr,nosuid,noquota,xattr,dev=9cec0043  1028939560
```

Figure 7.1: Per-Zone mnttab

Chapter 8

Networking

Consider a server which contains several zones as a result of a server consolidation program. Externally over the network, it will appear to be a multi-homed server which has inherited all the IP addresses of the original servers. However, internally it will look quite different from a traditional multi-homed server. The IP stack must partition the networking between the zones in much the same way it would have been partitioned between separate servers. Whilst the original servers could potentially all communicate with each other over the network, they could also all run the same services such as `sendmail(1M)`, `apache(1M)`, etc. The same features are provided by zones—the zones can all communicate with each other just as though they were still linked by a network, but they also all have separate bindings such they can all run their own server daemons, and these can be the same as those running in another zone listening on the same port numbers without any conflict. The IP stack resolves these conflicts by considering the IP addresses incoming connections are destined for, which identify the original server, and now the zone, the connection is considered to be in.

8.1 Partitioning

The IP stack in a system supporting zones implements the separation of network traffic between zones. Each logical interface on the system belongs to a specific zone (the global zone by default). Likewise, each stream/connection belongs to the zone of the process which opened it.

Bindings (connections) between upper layer streams and logical interfaces are restricted such that a stream may only establish bindings to logical interfaces in the same zone. Likewise, packets from a logical interface can only be passed to upper layer streams in the same zone as the logical interface. Applications which bind to `INADDR_ANY` for receiving IP traffic are silently restricted to receiving traffic from the same zone. Each zone conceptually has a separate set of binds (mainly used for listens), so that each zone can be running the same application listening on the same port number without binds failing due to the address already being in use. Thus each zone can, for example, run its own `inetd(1M)` with a full

configuration file, `sendmail(1M)`, `apache(1M)`, etc.

Each zone conceptually has its own loopback interface, and bindings to the loopback address are kept partitioned within a zone. An exception is the case where a stream in one zone attempts to access the IP address of an interface in another zone - such bindings are established through the pseudo loopback interface, as is currently the case in Solaris systems prior to zone support. As there is currently no mechanism to prevent such cross-zone bindings, existing Solaris firewalling products will not be able to filter or otherwise act on cross-zone traffic, as it is handled entirely within IP and is not visible to any underlying firewalling products. In the future as part of another project, an option might be provided to prevent such cross-zone bindings.

Sending and receiving broadcast and multicast packets is supported in all zones. Inter-zone broadcast and multicast is implemented by replicating outgoing and incoming packets as necessary so that each zone which should receive a broadcast packet or has joined a particular multicast group receives the appropriate data.

One other complication is the TCP global queue, `tcp_g_q`. When packets are sent up from IP to TCP, the only context available is the stream. Since the `tcp_g_q` isn't associated with any particular zone, the zone context is lost here. To get it back, a dummy IPsec `M_CTL` message is prepended to the IP packet to hold the zoneid. Note that for IPsec in and out, the zone context is preserved, and this is handled normally in the `M_CTL`.

Zones (other than the global zone) get restricted access to the network. The standard TCP/UDP transport interfaces are available, but some lower level interfaces are not. These restrictions are in place to ensure a zone cannot gain uncontrolled access to the network, such that it might be able to behave in undesirable ways (e.g., masquerade as a different zone or interfere with the network structure or operation, or obtain data from the network which does not relate to itself, etc.).

The `if_tcp(7P)` `SIOCTMYADDR` ioctl tests if a specified address is part of this node. The uses of this ioctl identified so far require a "node" to be interpreted as a zone.

8.2 Interfaces

Each zone which requires network connectivity will have one or more dedicated IP addresses. These will be associated with network interfaces. Typically, logical interfaces will be used for zones. Interfaces are placed in a zone by `ifconfig(1M)` using a new zone argument. This is done by use of a new `if_tcp(7P)` ioctl, `SIOCSLIFZONE` (and corresponding `SIOCGLIFZONE` to read back the value). Interfaces can only be configured from within the global zone; zone administrators will not be permitted to change the configuration of their network interfaces. In principle, zone administrators could safely be given access to bring their zone interface(s) up or down, but no requirement for this is identified and it will not be implemented in this project.

Within a zone, only that zone's interfaces will be visible to `ifconfig(1M)`, and via the existing `if_tcp(7P)` ioctls, `SIOCGLIFCONF` and `SIOCGLIFNUM`. In the global zone, `ifconfig(1M)`

```
# ifconfig -a
lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
hme0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 129.146.126.89 netmask ffffff00 broadcast 129.146.126.255
        ether 8:0:20:b9:37:ff

# ifconfig -aZ
lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
lo0:1: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1 zone n
ebbiolo-z1
        inet 127.0.0.1 netmask ff000000
hme0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 129.146.126.89 netmask ffffff00 broadcast 129.146.126.255
        ether 8:0:20:b9:37:ff
hme0:1: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2 zone
nebbiolo-z1
        inet 129.146.126.203 netmask ffffff00 broadcast 129.146.126.255

# zlogin nebbiolo-z1 ifconfig -a
lo0:1: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
hme0:1: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 129.146.126.203 netmask ffffff00 broadcast 129.146.126.255
```

Figure 8.1: Per-Zone Network Interfaces

can be run with a new `-Z` flag which applies the command to interfaces in all zones. Figure 8.1 shows an example where interfaces in all zones are listed. Note that as `ifconfig(1M)` currently contains an undocumented `-Z` flag which is used to trigger debugging, this will be changed to `-X` but remain undocumented. Finally, the `struct lifconf` used by `SIOCGLIFCONF` and the `struct lifnum` used by `SIOCGLIFNUM` include a new flag `LIFC_ALLZONES` which can be used to request interfaces in all zones be returned in the response. This flag is ignored if the requester is not in the global zone.

8.3 IPv6

As with IPv4, the use of IPv6 within a zone can be supported by placing logical interfaces in the zone. IPv6, however, does include a number of unique features (such as address auto-

configuration) which are discussed below that require special consideration when configuring them for use with zones. At the present time, support for IPv6 within non-global zones is planned to be part of a future project.

8.3.1 Address Auto-configuration

Unlike IPv4 where the global administrator will need to assign addresses to a zone, the use of the address auto-configuration feature of IPv6 provides a useful mechanism to generate unique addresses for the zone. Since typically the system's IEEE 802 48-bit MAC address is used to generate unique addresses for the global zone, a different mechanism is required for each of the local zones so that each has a unique EUI-64 interface identifier as described in RFC 2373 [19]. The existing IPv6 address token mechanism which was introduced as part of the IPv6 Protocol Stack and Utilities project (PSARC/1997/184) [3] can be extended to permit multiple tokens to be assigned to a physical interface, each tied to an associated zone (in the existing system, only a single token is permitted for a physical interface) and a per-zone token can be one of the properties that can be set for a network resource that has been assigned to a zone. When `in.ndpd(1M)` (which runs within the global zone) performs address auto-configuration, it can use the list of tokens assigned to a physical interface and the prefixes being advertised on that interface in order to plumb logical interfaces and assign them to their respective zones. It may also be possible to automate the generation of the interface identifier by combining part of the system's MAC address with the zoneid itself.

8.3.2 Address Selection

The IPv6 Default Address Selection project [31] introduced a mechanism for a global administrator to select which source and destination IPv6 addresses should be used when sending datagrams in the case where multiple addresses are available. Virtualizing this mechanism so that each non-global zone could individually specify their own source selection and destination ordering rather than inheriting the default settings from the global zone is under investigation.

8.3.3 6to4 Router

The creation of IPv6 tunnels (including the variety introduced by the 6to4 Router project (PSARC/2001/471) [15]) must be configured from within the global zone but they can be assigned to any zone for its use.

8.4 IPQoS

IPQoS enables IP traffic to be classified (for differential services, and/or accounting) according to a number of parameters, some obtained from the IP packet itself, some obtained from

the environment/context in which it appears (such as logical interface, uid, projid). IPQoS can only be configured by the global administrator from within the global zone.

It is anticipated that people may want to be able to classify IP traffic by zone. Since each zone has one or more unique IP addresses, the IP address can be used to do this.

A future project could be the addition of a zone classification parameter. However, this will have an impact on the performance of the IP fast-path when IPQoS is operating. This effect will become quite significant if, for example, an existing single system configuration is simply copied once for each zone resulting in many more filters to match. Such an enhancement is not proposed at this time, but could be reviewed in the future.

The flow accounting module `flowacct(7IPP)`, when recording extended attributes, includes the uid and projid within flow accounting records. These are not necessarily meaningful without the zoneid. The zone can be derived from the IP address also included in the records.

In a future project, the zoneid could be added to the flowacct 8-tuple to form a 9-tuple for flow identification purposes.

8.5 IPsec

IPsec configuration applies system-wide, but zone-specific configuration can be created by specifying a zone's IP address as the `laddr` field in a ruleset. Tunnels, just like logical interfaces, can be placed into non-global zones, and used with IPsec. IPsec operation between zones will be the same as that currently operated over the loopback interface—there are some performance gains made here on the basis that the traffic is not exposed on any external interfaces.

IPsec can be configured only from the global zone.

8.6 Raw IP Socket Access

General raw IP socket access will not be available in a non-global zone. Such access gives a privileged user in the zone the uncontrolled ability to fabricate and receive packets contrary to the network partitioning between zones. Because `traceroute(1M)` requires the use of raw IP sockets to generate IPv4 datagrams, the use of this command in a non-global zone will not be supported in this project.

One special case of raw socket access which will be supported is for the ICMP protocol as this is required by the `ping(1M)` command. To support this, the introduction of a new privilege is under consideration for the ability to send and receive appropriate ICMP messages by a privileged user from within a non-global zone.

8.7 DLPI Access

DLPI access provides the raw interface to the network drivers on Solaris. There is no requirement for this feature from within a zone. Until a requirement for DLPI access is identified, no DLPI access will be provided from within a zone. Such a requirement might result from the desire to run `snoop(1M)` in a zone, but this is a complex area which will need some further thought, particularly how to limit snoop to show only link layer packets related to the caller's zone.

In current Solaris systems without zone support, DLPI access is restricted using the `PRIV_NET_RAWACCESS` privilege, introduced by PSARC/2002/188 [7] (see Section 5.2). The `PRIV_NET_RAWACCESS` privilege would not be available inside a zone.

8.8 Routing

Routing remains a system-wide feature, just as it is today on Solaris systems prior to zone support. Since routing changes affect the whole system, routing changes will only be allowed from the global zone. Views of the routing table from within zones will be restricted to routes relevant to that zone.

8.9 IP Multipathing

IP Multipathing will continue to work in a systems containing zones. In the event a physical interface fails, the logical interfaces on it (which potentially belong to different zones) will be transferred to a backup physical interface. IP Multipathing can only be configured from the global zone.

8.10 Mobile IP

Mobile IP depends on making configuration changes which have system-wide implications, and therefore it cannot be used inside a zone where such changes are not permitted.

8.11 NCA

NCA will not be available to web servers within a non-global zone.

8.12 DHCP

DHCP could potentially be used by the system owner to administer IP addresses for the zones. In order to do this, it would be necessary to teach DHCP how to acquire addresses

for interface aliases. This would require some work in that the DHCP client side currently makes some assumptions about the identifier that is used for allocating an address, so further investigation would be needed. This is not planned at this time, as in the context of server consolidation, servers do not normally obtain their IP addresses via DHCP.

8.13 Tuning

System level tuning parameters set via `ndd(1M)` or `/etc/system` will not be configurable from within a zone. `ndd(1M)` parameters may be read from within a zone. Where relevant, the kernel only provides ndd with data relating to the caller's zone, e.g., when listing connection details, IRE entries, etc.

8.14 Unbundled Software

8.14.1 SunScreen

It is not envisaged that this project will cause any problems with SunScreen, although it now seems unlikely SunScreen will be supported in Solaris 10 anyway.

8.14.2 IP-Filter

IP-Filter is a very popular freeware firewalling and NAT package on Solaris (and several other versions of Unix). A project is currently underway to integrate this into Solaris 10, PSARC/2003/046 [23].

IP-Filter, as currently structured, will not be able to filter any IP traffic between zones, as this is handled directly in IP and is not visible to IP-filter. It would be advantageous to be able to do this, but this is something which will not be part of this project. The project team and the Solaris IP Filter project team have talked about the possibility of implementing this, and it could form part of a future project.

8.14.3 CheckPoint Firewall

No problems are anticipated operating with CheckPoint Firewall or any other firewall product which is pushed onto the NIC device drivers. Administration and control would only be available from the Global Zone.

8.14.4 Siemens-Nixdorf CMX 5.0 Transport Gateway

We have a contract with Siemens-Nixdorf to provide a private interface `IP_ADD_PROXY_ADDR` for their CMX 5.0 Transport Gateway. This will not be made available inside a non-global zone.

8.14.5 Sun Cluster

The Sun Cluster contracted ioctl `TCP_IOC_ABORT_CONN` (PSARC/2001/292) [5] will not be supported in a non-global zone.

Chapter 9

Devices

All applications make use of devices; however, the great majority of applications only interact directly with pseudo-devices, which makes the task of providing a zoned device environment feasible. The key goals for providing devices in a zone are:

Security: Users interacting with devices appearing in a zone must not be able to use those devices to compromise another zone or the system as a whole.

Virtualization: Some devices must be modified to provide namespace or resource isolation for operation in a zone.

Administration: It must be easy to place a default, safe collection of devices in a zone; warnings must occur when an administrator attempts to place unsafe devices into a zone; it must be possible for a knowledgeable administrator to assign physical devices to zones when needed.

Automatic Operation: With the advent of `devfsadm(1M)` [13], device file system management in Solaris became largely automated. Zones should not require administrator intervention to create dynamically managed device nodes.

This chapter begins by classifying devices according to their virtualization and security characteristics. Discussions of the `/devices` namespace, device privilege and permission, device administration tools, and the special handling required to support pseudo terminals follow.

9.1 Device Categories

Treatment of devices with respect to zones must of necessity vary depending on the type of device. For the purposes of this discussion, devices can be divided into the following categories:

1. *Unsafe*: Devices that cannot be safely used within a zone.
2. *Fully-Virtual*: Devices that reference no global state, and may safely appear in any zone.
3. *Sharable-Virtual*: Devices that reference global state, but may be safely shared across zones, possibly as the result of modification made by this project.
4. *Exclusive*: Devices that can be safely assigned to and used exclusively by a single zone.

9.1.1 Unsafe Devices

Examples of unsafe devices include those devices which expose global system state, such as:

- `/dev/kmem`
- `/dev/cpc` (`cpc(3CPC)`)
- `/dev/trapstat` (`trapstat(1M)`)
- `/dev/lockstat` (`lockstat(7D)`)

There is no way to allow use of such devices from a zone without violating the security principles of zones. Note that this is not restricted to control operations; for example, read access to `/dev/kmem` will allow a zone to snoop on activity within other zones (by looking at data stored in kernel memory).

This category includes most physical device instances present on the platform, including bus nexus devices, platform support drivers, and devices in support of the device administration infrastructure. All of these devices are central to the operation of the platform as a whole, and are not appropriate to expose for monitoring or control by the lower-privilege environment inside a non-global zone.

9.1.2 Fully-Virtual Devices

A few of the system's pseudo devices are "fully virtualized"; these are device instances that reference no global system state, and may safely appear in any zone. An excellent example is `/dev/tty` (`tty(7D)`), which references only the controlling terminal of the process in whose context it executes.

Other fully-virtual devices include:

- `/dev/null` (`null(7D)`) and `/dev/zero` (`zero(7D)`).
- `/dev/poll` (`poll(7D)`)
- `/dev/logindmux`, used to link two streams together in support of applications including `telnet(1)`.

9.1.3 Sharable-Virtual Devices

Those device instances which reference some sort of global state, but which may be modified to be zone-compatible are said to be *Sharable Virtual Devices*. Some examples include:

- `/dev/kstat` (`kstat(7D)`)
- `/dev/ptmx` (`ptm(7D)`), the pty master device. See the "Pseudo-Terminals" discussion in Section 9.8.

A principal example of such a device is the `random(7D)` driver, which exports the `/dev/random` and `/dev/urandom` minor nodes. In this case, the global state is the kernel's entropy pool [27], from which it provides a stream of cryptographic-quality random bytes.

9.1.4 Exclusive-Use Devices

We expect that many customers will have a small number of devices which they wish to assign to specific zones. The chief among these is likely to be SCSI tape devices, although a better solution would be to employ a network backup solution such as the Legato Networker client.

There are numerous problems with allowing exclusive-use devices to be accessed by zone users. These are not really significantly different from Solaris's limitations today with respect to allowing unprivileged users to access specific devices; it may be possible for a user to mistreat a device in such a way that system panic, bus resets, or other adverse effects occur. This may be seen as a fundamental limitation of the zoned approach to isolation and virtualization. A thorough treatment of the problems surrounding SCSI devices is outside the scope of this discussion, and is discussed in detail in the `sgen(7D)` manual page and PSARC/1999/525 [30].

A slightly more subtle problem may occur when a non-virtualized device assigned to a zone is also used by unwitting global zone applications. And example is a disk device which has been shared into a zone. If both a non-global zone application and a global zone application independently begin writing to the disk, both may face data corruption. While such devices (such as disks or tapes) are semantically designed for "exclusive use," there is no kernel-level mechanism in Solaris to enforce such a usage pattern beyond the `O_EXCL` option to `open(2)`; the project does not propose to add such a mechanism. To summarize: administrators placing a physical device into more than one zone risk creating a covert channel between zones; furthermore, global zone applications using such a device risk compromise or data corruption by a hostile zone.

9.2 /dev and /devices Namespace

The `devfs(7FS)` file system is used by Solaris to manage `/devices`. Each element in this namespace represents the physical path to a hardware device, pseudo device, or nexus device; it is a reflection of the device tree. As such, the file system is populated by a hierarchy of directories and device special files.

The `/dev` file hierarchy, which is today part of the / (root) file system, consists of symbolic links (logical paths) to the physical paths present in `/devices`. `/dev` is managed by a complex system comprised of `devfsadm(1M)`, `syseventd(1M)`, the `devinfo(7D)` driver, `libdevinfo(3LIB)` and RCM.

Few end-user applications reference `/devices`, and the stability of file path names in this file system appears to be undefined from release to release of Solaris. Instead, applications reference the *logical path* to a device, presented in `/dev`. So, while the system's `/devices` file system is important to a few system administration applications, there is no clear mandate that it must appear in a zone. After consultation with the Solaris I/O organization, we propose to `mknod(2)` the appropriate device special files in `/dev` in a zone via new extensions to `devfsadm(1M)` and omit `/devices` entirely.

It is important to note that some Solaris engineers have voiced objection to this limitation. Some contributors believe that it is "better" to use a device from the `/devices` path than the `/dev` path; part of this seems to be an unwillingness to expose devices in `/dev` which customers may regard as interfaces, and part seems to be folklore. The project team believes that advice should be given to all project teams to make sure a link is created in `/dev`, even for a device with a private interface. However, the project team believes that the interface concern is legitimate; to resolve it, the project team notes that a `/dev/SUNWprivate` directory could be mandated by the ARC for use by devices not exporting publicly visible interfaces; this would have the side-effect of cleaning up some crufty entries in `/dev`.

9.3 Device Management: Zone Configuration

An interface is provided to indicate which devices should appear in a particular zone; this functionality is provided by the `zonecfg(1M)` infrastructure using a flexible rule-matching system. Devices matching one of the rules are included in the zone's `/dev` file system. Some elements of the default set of device rules are shown:

```
zonecfg> info device
device: pts
        matchtype: devpath
        match: /dev/pts/*
device: ptmx
        matchtype: devpath
        match: /dev/ptmx
```

```
device: logindmux
        matchtype: drv_min
        match: clone:logindmux
device: lo
        matchtype: driver
        match: lo
...
```

To include an additional device in the zone, the administrator may specify it by adding an additional rule:

```
zonecfg> add device my_scanner
zonecfg> setprop device my_scanner matchtype devpath
zonecfg> setprop device my_scanner match /dev/scsi/scanner/c3t4*
zonecfg> info device my_scanner
device: my_scanner
        matchtype: devpath
        match: /dev/scsi/scanner/c0t0*
```

Using this syntax, a device may specified in one of three ways; the type of matching used is determined by the *matchtype* property.

- *Device Path (devpath)*: This is expected to be the most common way to specify a device. The candidate device's global zone /dev pathname is tested against the *match* property using `fnmatch(3C)`.
- *Driver Name (driver)*: The driver name of the candidate device for the zone is tested against the *match* property.
- *Driver Name & Minor Name (drv_min)*: Both the driver name and minor name of the candidate device are compared to the *match* property, which is expected to be in the format *drivername:minorname*.

It is worth noting that this matching system is extensible and could be extended to match by Fibre-Channel WWN or by any other device naming scheme devised in the future.

9.4 Device Management: Zone Runtime

The project adds new consolidation-private interfaces and capabilities to the the `devfsadm` daemon; the changes are somewhat obscure, and are enumerated below:

1. When the daemon starts up, it discovers which zones on the system are READY or RUNNING; it is assumed that such zones have a valid /dev file hierarchy. `devfsadmd` retains a list of said zones; it also loads the zone's configuration database in order to know the `<device>` matching rules.

2. When a zone enters the READY state from the INSTALLED state, the `zoneadm` tool invokes the `devfsadm` command (not the daemon) and passes arguments to it indicating which zone's `/dev` directory should be populated. Additionally, the zone is registered with the `devfsadmd` daemon, via a door call to a new door, `/dev/.zone_reg_door`.

3. When a zone is discovered by (1) or (2) above, `devfsadmd` creates a file under the zone's root directory at `/dev/.devfsadm_synch_door` [18] and attaches the appropriate door to it. See Section 9.8 for more information. It also loads the zone's configuration database in order to know applicable `<device>` matching rules. If the zone being registered is *already* registered, its configuration is re-read, and its door file is re-created.

4. When a zone enters the INSTALLED state from the READY state, the zone is unregistered from `devfsadmd`, which then frees associated resources, and ceases to manage it.

5. When a device-related event occurs (for example in response to a hotplug event), a device entry in `/dev` may need to be created. When this occurs, the `devfsadmd` link-generator module calls the `devfsadm_mklink()` routine.

   This routine first services the global zone, establishing the device symbolic link requested by the link-generator. Next, the routine iterates across all registered zones. Instead of creating links, however, device nodes are created with mknod(2); node creation is subject to the filtering rules for the zone's configuration.

A significant problem with this approach is that `devfsadmd` must "reach into" zones in order to mknod, create, delete and symlink files. This violates zone file system principles outlined in Chapter 7, and there is a distinct danger of buffer-overrun and symlink attacks. The project team is currently studying how best to address this problem. One possible solution is that `devfsadmd` could employ a delegate process which could enter the zone with sufficient privilege to manipulate the `/dev` hierarchy. Unfortunately, an attack against this delegate could be similarly devastating if it could be forced to mknod something other than its intended node. Another possibility is to loopback mount the `/dev` file system into the zone; device nodes could be freely accessed (opened `O_RDWR`, for example) by zone processes, but other file operations (creation, unlink, symlink, link, etc.) would be prevented. This is similar to the historical operating mode of the `lofs(7FS)` file system when mounted in read-only mode [22].

9.5 Device Management: Future Work

The point must be emphasized: we propose to run a single instance of `devfsadmd` system-wide; however, this does not preclude a more advanced architecture for managing zoned devices in the future. This approach, while not entirely elegant, was chosen for its simplicity and because none of the alternatives examined were more palatable.

We envision that the advent of a file system capable of managing the /dev namespace will make the management of /dev both in the global and non-global zones significantly simpler. Some initial discussion has occurred about the creation of such a file system with the Solaris I/O organization.

9.6 Device Privilege Enforcement

In Solaris, privileges held by applications interact with device drivers in a complex way. File system permissions, device policy (see `getdevpolicy(1M)`) and driver-based privilege checks (`drv_priv(9F)` and `priv_policy(9F)`) may all come into effect during a call to `open(2)`. This section discusses issues related to driver privileges.

9.6.1 DDI Impact

With the introduction of fine-grained privileges, calls to `drv_priv(9F)` will succeed if the calling process has the `PRIV_SYS_DEVICES` privilege. This privilege also allows the creation of device nodes, so it is not safe within a non-global zone. Thus, calls to `drv_priv(9F)` by processes within non-global zones will fail. This prevents potentially unsafe driver operations, but may be too restrictive in some cases; a driver may wish to restrict an operation to privileged users, but allow it for such users within a non-global zone. One way to accomplish this would be to check for a privilege that is available within non-global zones. The project team is currently investigating whether this is sufficient, or whether some additional interface is needed.

9.6.2 File System Permissions

Of particular concern is the common use of file system permissions to restrict access to devices. In a zone environment, where a process with super-user privileges may not be trusted, such controls are not useful as a way of enforcing safe device use. For devices where some operations are unsafe, checks to prevent those operations by unprivileged processes (including any processes in a zone) must be implemented in the kernel, not as permissions on the device node.

A good example is the `random(7D)` driver:

```
$ ls -lL /dev/*random
crw-r--r--   1 root     sys      190,  0 Oct 15  2001 /dev/random
crw-r--r--   1 root     sys      190,  1 Oct 15  2001 /dev/urandom
```

In the global zone, a sufficiently priveleged user can write to this device; the `random` driver code contains no privilege checks of any kind. In a non-global zone, however, no process should be able to write to this device. In this case, the code for the `random` driver must be modified to perform privilege enforcement in the kernel.

9.6.3 The Trouble with Ioctls

Some drivers perform fine-grained privilege checks using `drv_priv(9F)` or `priv_policy(9F)` when file system permissions are not sufficient to express access control. A common occurrence is an ioctl handler in which some subcodes require elevated privilege to execute.

For a driver using fine grained checks to be safe for use in a zone:

- Each such check must be examined by a developer for its safety in a zoned environment. For those ioctls which are safe, appropriate privilege checks may be added.
- Each ioctl which might impact long term device state or platform hardware state must be evaluated.
- New privilege checks may need to be added for ioctls deemed unsafe.

9.6.4 Illegal dev_t's

A significant security concern is that a privleged zone user must not be able to "import" character or block devices into the zone from NFS mounts, lofi mounted devices, or by any other means. This is because a rogue special file introduced into the zone could contain the `dev_t` for `/dev/kmem` or other privileged devices. This would provide a privileged user within the zone with undue power.

One proposal for mitigating this problem is to introduce a new pair of mount options, such as `-o devices`/`-o nodevices`, which allow a file system to be mounted such that all open()s of special files would fail (this is actually a subset of the functionality provided by the `-o nosuid` option today). This is under investigation, and is considered a requirement for this project.

9.7 Device Driver Administration

Most operations concerning kernel, device and "platform" management will not work inside a non-global zone, since modifying platform hardware configurations violates the zone security model. These operations include:

- Adding and removing drivers.
- Explicitly loading and unloading kernel modules (distinct from the implicit loads which can happen as a result of `I_PUSH` and other indirect operations).
- Using DR initiators (i.e., `cfgadm`).
- Facilities which affect the state of the physical platform.

The following operations should work inside a non-global zone:

- Examining the list of loaded kernel modules (i.e., `modinfo(1M)`).

Ideally, the project team would like to also provide the use of `prtconf(1M)`, `libdevinfo(3LIB)` (i.e., the `devinfo(7D)` driver in read-only mode), and the read-only `ioctl`s exported by the `openprom(7D)` driver (enabling `prtconf -p`). However, support for these elements of the device framework are not seen as critical requirements for the project at this time.

Based upon feedback from internal "alpha" users, the project team has investigated supporting `prtdiag(1M)`. However the architecture of `prtdiag` is so poorly factored that supporting the platform-specific information normally exported by it is almost totally infeasible. Extensive effort and testing would be required to support the 13 "platforms" currently supported by `prtdiag`, and would require zones to support a host of sun4u platform-specific subsystems including `picld(1M)`, `envctrl` and others. Some of the platform code in `prtdiag` also relies on `/devices` as an interface.

As an alternative, the project team is investigating (but not committed to at this time) writing a "generic" plugin for prtdiag which would print some minimal system information using exclusively public interfaces available on all platforms. A pleasant side-effect of this work would be a minimally capable `prtdiag(1M)` implementation for the IA32 architecture. This may be postponed and considered as future work.

9.8 Pseudo-Terminals

Solaris's pseudo-terminal support is comprised of a pair of drivers, `ptm(7D)`, and `pts(7D)`; several STREAMS modules, including `ptem(7M)` (terminal emulator), `ldterm(7D)` (line discipline), and `ttcompat(7M)` (V7, 4BSD and XENIX compatibility); and userland support code in `libc` and `/usr/lib/pt_chmod`.

The `ptm` and `pts` drivers are enhanced such that an open of a `pts` device can only occur in the zone which opened the master side for the corresponding instance (the ptm driver is self-cloning).

In the case of the `zlogin(1)` command, it is necessary to allocate a pty in the global zone, and to then "push" that pty into a particular zone. To accomplish this, a private `zonept(3C)` library call is introduced. `zonept` issues the `ZONEPT` ioctl to the master device requesting that the current terminal be assigned the supplied zone owner.

The number of `pts` devices may grow without bound[1]. This dynamic growth is triggered when the number of ptys must grow beyond the current limit (for example, when allocating the 17th pty). This must function even when the application opening `/dev/ptmx` is unprivileged. This functionality (provided in `libc`) relies on a door call to `devfsadmd`, which is wrapped by the `di_devlink_init()` interface. The code can also start `devfsadmd` as needed.

[1] In practice, the number of `pts` devices on the system is limited by the size of the minor number space in the operating system.

To solve this problem in the zone, we place the appropriate door to the global zone's `devfsadmd` into the non-global zone. This allows the zone to demand that the global zone install the appropriate `/dev/pts/` device nodes as needed. There is some risk of denial-of-service attack against `devfsadmd` here, and there exists the possibility that `devfsadmd` is not running. We are currently investigating solutions to these problems. One promising possibility is to allow the `zparent` process to act as a "proxy server" for such door calls; `zparent` would simply turn the zone's request for devlink creation into a call to `di_devlink_init()`, which would in turn start `devfsadmd` if it was not running. `zparent` could throttle requests as needed.

A final issue is that it is desirable for the global administrator to be able to limit the number of pseudo-terminal devices available to each zone. One possibility is to implement a zone-scoped resource control (See Chapter 10) for pty creation. This is addressed at least in part by RFE 4630671, but is not considered a dependency for this project.

9.9 /dev/kstat

`/dev/kstat` is used extensively for applications monitoring the system's performance including `mpstat(1M)`, `vmstat(1M)`, `iostat(1M)` and `kstat(1M)`. Unlike traditional Solaris systems, the values reported in a particular kstat may not be relevant (or accurate) in non-global zones.

Statistics fall in to one of the following categories as far as zones are concerned:

1. Those that report information on the system as a whole, and should be exported to all zones. Most kstats currently fall under this category. Examples include *kmem_cache* statistics.

2. Those that should be virtualized: these kstats have the same module:instance:name:class but export different values depending on which zone is viewing them. Examples include *unix:0:rpc_client* and a number of other kstats consumed by `nfsstat(1M)` that should report virtualized statistics since the subsystems they represent have been virtualized.

3. Those that "belong" to a particular zone and should only be exported to the zone to which they "belong", (and in some cases, the global zone as well). *nfs:*:mntinfo* is a good example of this category, since zone A should not be exported information about the NFS mounts in zone B.

The kstat framework has been extended with new interfaces to specify (at creation time) which of the above classes a particular statistic belongs to.

It may make sense to only export device statistics for the devices which exist within a zone; thus, if a particular device is completely invisible from a zone, it would make sense to not export its statistics to the zone. Statistics that fall into this category include those for network interfaces, disks, and cpus. Exporting these statistics, however, does not necessarily

allow untrusted processes in a zone to obtain undue information about activity in other zones.[2]

[2]There has been talk about limiting the list of cpus exported to a zone based on the zone's resource pool binding, but this idea has not yet been fleshed out.

Chapter 10

Resource Management and Accounting

One area of isolation not directly addressed by zones is that of resource usage. Processes in one zone could consume CPU, physical memory, or swap space at the expense of processes in other zones.

We plan to address this problem in two ways. One is by preventing processes within a zone, regardless of effective uid, from performing operations that unfairly increase the resources they are allocated. In general, these correspond to the operations that require super-user privileges when running in the global zone. For example, processes inside a zone will not be able to assign processes to the real-time scheduling class, create processor sets, or lock down memory. The restrictions on privileges are listed in Section 5.2.

Note that in some configurations such operations can be safe. If the processes assigned to a zone are also assigned to a processor set (and no other processes are assigned to the processor set), then the processes in the zone can freely control scheduling priority (including creating real-time processes) without affecting other processes in the system. Likewise, it may be safe to allow locked memory if a memory set has been created that exactly matches the boundaries of the zone. By coupling zones with a sufficiently fine grained privilege mechanism, an administrator could allow a specific zone to have privileges such as scheduling control not allowed within other zones.

This leads to the second method of addressing resource usage between zones. Just as resource management technologies like a fair-share scheduler [8] or resource pools [17] can be used to control the resource usage of different applications running on the same system, they can similarly be used to control the utilization of different zones. In fact, it becomes advantageous to align the boundaries of resource management controls with those of zones; this leads to a more complete model of a virtual machine, where namespace access, security isolation, and resource usage are all controlled.

In this chapter, we discuss the ways in which the Solaris resource management features should interact with the zone functionality. We start by discussing the existing features and how they will be modified or used with zones.

10.1 Solaris Resource Management Interactions

The resource management features in Solaris and their interactions with zones are summarized below:

10.1.1 Accounting

10.1.1.1 System V Accounting

The traditional accounting system uses a fixed record size, and can not be extended to differentiate between a process running in the global zone and a non-global zone. We plan to modify the system such that accounting records generated in any (including the global) zone only contain records pertinent to the zone.

10.1.1.2 Extended Accounting

The extended accounting subsystem is virtualized to permit different accounting settings and files on a per-zone basis for process- and task-based accounting. Since exacct records are extensible, they can be tagged with a zone id (EXD_PROC_ZONEID and EXD_TASK_ZONEID, for processes and tasks, respectively), allowing the global administrator to determine resource consumption on a per-zone basis. Accounting records are written to the global zone's accounting files as well as their per-zone counterparts. The EXD_TASK_HOSTNAME, EXD_PROC_HOSTNAME, and EXD_HOSTNAME records contain the `uname -n` value for the zone in which the process/task executed[1], rather than the global zone's node name. Flow accounting is covered in Section 8.4.

10.1.2 Projects, Resource Controls and the Fair Share Scheduler

Projects are abstracted such that different zones may use separate `project(4)` databases, each with their own set of defined projects and resource controls. Projects running in different zones with the same project id are considered distinct by the kernel, eliminating the possibility of cross-zone interference, and allowing projects running in different zones (with the same project id) to have different resource controls.

In order to prevent processes in a zone from monopolizing the system, we will introduce zone-wide limits applicable resources, limiting the total resource usage of all process entities within a zone (regardless of project). The global administrator will be able to specify these limits in the `zonecfg` configuration file[2]. Privileged zone-wide rctls can only be set by superuser in the global zone.

Currently planned zone-wide rctls include:

zone.cpu-shares Top-level number of FSS shares allocated to the zone. Cpu shares are thus first allocated to zones, and then further subdivided among projects within the zone

[1] Tasks may not span zones.

[2] The exact interface for specifying this has not yet been determined.

(based on the zone's `project(4)` database). In other words, `project.cpu-shares` is now relative to the zone's `zone.cpu-shares`. Projects in a zone will multiplex the shares allocated to the zone; in this way, FSS share allocation in a zone can be thought of as a two-level hierarchy.

zone.tmpfs-swap-limit This rctl would limit the total amount of swap consumable by tmpfs file systems. Having this rctl in place would let us allow tmpfs mounts from within a zone, but the details have not yet been fleshed out. We furthermore need a rctl to limit the amount of kmem consumable by tmpfs mounts in a zone, an issue that needs to be addressed regardless of whether tmpfs mounts are permitted in a zone. See Section 7.3 for more information about tmpfs in a zone.

zone.max-shm-memory, zone.max-shm-ids, zone.max-msg-ids, zone.max-sem-ids The idea is to provide zone-wide controls to complement the project-wide ones. Details TBD.

10.1.3 Resource Pools

When PSARC/2003/136 "libpool(3LIB) enhancements" it is planned to add an attribute to the zone configuration, `zone.pool` similar to `project.pool`. The zone as a whole will be bound to this pool upon creation, and it will be possible to enforce this binding in the kernel, as well as limit the visibility of resources not in the resource pool.

It is currently planned that non-global zones must be bound to resource pools in their entirety; that is, attempting to bind individual processes, tasks, or projects in a non-global zone to a resource pool will fail. This allows the virtualization of pools such that the `pool` device only reports information about the particular pool that the zone is bound to. The `poolctl` device will not be included in zones by default, hence zone administrators will not be able to change their pool binding or change the configurations of pools.

`poolstat(1M)`, described in PSARC/2003/137, will thus only reveal information about the resources the zone's pool is associated with.

10.2 Resource Observability within a Zone

In cases where resource allocation is aligned with zones, one could consider changing the behavior of interfaces that query the system about resource availability, restricting such interfaces to reporting information about the resources available within the zone rather than those of the system as a whole. For example, if the processes in a zone have all been assigned to the same processor set (or CPU set in a resource pool), the `psrinfo(1M)` utility could only display the processors within that set, and the `getloadavg(3C)` interface could report the load average of the set. This would provide applications and administrators within the zone with the information most relevant to them, rather than information about other zones in the system over which they have no control.

The problem with this approach is that the overall complexity of the relationship between resource allocation and zones. Since we don't want to prevent the use of zones on a uniprocessor system, we can't require that a specific set of processors be dedicated to each zone. This means that we can't always guarantee that it will be possible for `psrinfo(1M)` to report a list of processors relevant to the specific zone; even if the processes in the zone *are* all assigned to the same processor set, they may share that set with processes from another zone. The situation is even worse in a system using the fair-share scheduler to isolate zone activity; in that case, the processes within each zone receive a share of the overall system's CPU capacity, and it doesn't make sense to break that down by specific processors.

On a system with a 1-to-1 mapping between zones and resource sets, however, these statistics will make sense. At this point, the issue of which resource monitoring interfaces are to virtualized in a zone has not yet been agreed upon.

Chapter 11

Inter-Process Communication

Local inter-process communication (IPC) represents a particular problem for zones, since processes in different (non-global) zones should normally only be able to communicate via network APIs, as would be the case with processes running on separate machines. It might be possible for a process in the global zone to construct a way for processes in other zones to communicate, but this should not be possible without the participation of the global zone. In this chapter, we look at the different forms of IPC and how this issue will be handled for each.

11.1 Pipes, STREAMS, and Sockets

IPC mechanisms that use the file system as a rendezvous, like pipes (via fifofs), STREAMS (via namefs), and sockets (via sockfs), fit naturally into the zone model without modification since processes in one zone will not have access to file system locations associated with other zones. Since the file system hierarchy is partitioned, there is no way to achieve the rendezvous without the involvement of the global zone (which has access to the entire hierarchy).

11.2 Doors

Doors also use the file system as a rendezvous (via namefs), so potential door clients will normally only be able to call servers within the same zone. Doors, however, also provide a way of safely supporting cross-zone communication, since the server can retrieve the credentials of the caller using `door_ucred(3DOOR)`. We have extended the private data structure returned by `door_ucred` to include a zone id, and added a `ucred_getzoneid(3C)` interface to retrieve the zone id from the structure. This allows the creation of truly global door servers, if desired; a door served from the global zone could be mounted in each zone, and the server could check whether the caller is authorized to perform a given operation based on its zone id as well as other credential information. Although we feel that most such services should be written using networking interfaces for optimal flexibility and portability, this provides a

means for doing efficient cross-zone communication using doors when necessary. In fact, this functionality is used to implement the per-zone device node creation described in Chapter 9 (allowing processes in each zone to contact the global `devfsadmd` daemon).

11.3 Loopback Transport Providers

The loopback transport providers, ticlts, ticots, and ticotsord, provide yet another mechanism for IPC. These transports, like traditional network transports such as UDP or TCP, can be accessed using standard transport-independent TLI/XTI interfaces. However, the loopback transports are limited to communication between processes on the same machine and are implemented as pseudo devices without involving the kernel networking stack. The transport providers support "flex addresses", which are arbitrary sequences of octets of length greater than 0. When zones are in use, the flex address space will be partitioned to prevent communication between processes in different zones. In other words, each zone will have its own flex address namespace. This is done by internally (within the `tl` driver) associating zone ids with transport endpoints, based on the zone id of the process performing the `bind(3SOCKET)` call. This means that a process calling `connect(3SOCKET)` will only connect to the endpoint with a matching address associated with the caller's zone. In addition, multiple processes can bind to the same address, as long as they are in different zones; this means that multiple applications can use the same address without conflict if they are running in different zones, avoiding the need for cross-zone coordination in address selection.

11.4 System V IPC

The System V IPC interfaces allow applications to create persistent objects (shared memory segments, semaphores, and message queues) for communication and synchronization between processes on the same system. The objects are dynamically assigned numeric identifiers that can be associated with user-defined keys, allowing usage of a single object in unrelated processes. Objects are also associated with an owner (based on the effective user id of the creating process unless explicitly changed) and permission flags that can be set to restrict access when desired.

In order to prevent sharing (intentional or unintentional) between processes in different zones, a zone id is associated with each object, based on the zone in which the creating process was running at time of creation. Processes running in a zone other than the global zone will only be able to access or control objects associated with the same zone. There will be no new restrictions for processes running in the global zone (though the existing user id based restrictions will be honored); this allows an administrator in the global zone to manage IPC objects throughout the system without needing to enter each zone. The key namespace will also be made per-zone; this avoids the possibility of key collisions between zones.

The administrative commands, `ipcs(1)` and `ipcrm(1)`, have been updated with zone-

specific options for use when run in the global zone. By default, `ipcs` will report objects from the current zone. When run in the global zone, however, the `-z` *zone* option can be used to report objects from the specified zone, and the `-Z` option can be used to report objects from all zones, with the zone association of each identified. This can be used to disambiguate between objects in different zones which may have identical keys, and in general to provide better observability into the usage of IPC objects within zones.

The `ipcrm` command similarly operates on objects in the current zone, unless run in the global zone and given the `-z` *zone* option. This option allows removal of objects in other zones.

In the past, the System V IPC implementation in Solaris included a number of static system-wide limits on the number of objects that can be created, as well as various details of those objects (maximum size of shared memory segments, maximum depth of message queues, etc.). In a system with zones, this would lead to an obvious problem with processes in one zone exhausting all of the available objects. Fortunately, this implementation was recently revised to eliminate the need for these limits [29]; the only tunable parameters in the new code are per-project and per-process *resource controls* [16], which require no adaptation to work in a zone environment. We are also planning to implement some zone-wide resource controls to prevent (or at least allow administrators to avoid) kernel memory exhaustion by one zone. These are described in Section 10.1.2.

11.5 POSIX IPC

The POSIX IPC interfaces implemented in `librt`, unlike the System V interfaces, use files to rendezvous between processes. This means that they fall into the same category as pipes, sockets, and streams, in that no changes are necessary to enforce per-zone isolation and object key namespaces.

11.6 Event Channels

The introduction of General Purpose Event Channels [20] represents yet another IPC mechanism in Solaris. Event channels are similar to the loopback transports or System V IPC in that inter-process rendezvous is established through use of a key (a string representing the channel name) provided by participating processes. As with the other key-based IPC mechanisms, cross-zone communication will be prevented by providing each zone with a separate, independent namespace.

Event channels do present some unique challenges, however. Since each event channel requires a significant amount of kernel memory, some mechanism for controlling the number of channels created in each zone may be required. In addition, the ability for certain event channels to be allowed to span zones may be required. This is because event channels are expected to be used as part of the fault management architecture for Solaris [32]; it is

expected that certain types of events will need to be communicated from either the kernel or the global zone to non-global zones. The issues here are currently being investigated; it is expected that the ability to communicate between zones using event channels will be restricted to the specific channels used to communicate fault and error events, and may require use of a private API.

Chapter 12

Install and Upgrade

In order for processes in a non-global zone to be able to access files delivered as part of Solaris, there needs to be a mechanism to lay down a root file system for the non-global zone in such a way that the standard Solaris packaging tools can be used to administer the file system both by the global administrator and perhaps even by the zone administrator of the non-global zone. Examples of the former might including upgrading the system to a new version of Solaris (necessitating changes in both the global and non-global zones) while the latter might be a zone administrator adding a package for a unbundled or third-party product or applying a patch.

12.1 File Access

As discussed in Chapter 7, zones require a separate root file systems for isolation. In many cases, a non-global zone only requires a minimal amount of storage. For example, it requires a `/dev` directory for `devfsadm(1M)` to populate device nodes as well as `/proc` and `/tmp` mount points in order to the mount standard procfs and tmpfs file systems. It also requires a `/etc` directory to hold standard configuration information for that zone and a `/var` directory to contain both runtime and long-term state.

It is also highly desirable to allow read-only sharing of file system resources when that sharing by a non-global zone does not compromise the integrity of the global zone or any other zone on the system. An example of such a resource is shared libraries and executables that can be used by processes across all zones which results in more efficient use of the virtual memory system.

Although it's certainly possible to create a non-global zone that has an arbitrary mixture of local and shared file system objects, such zones are difficult to maintain and upgrade. The Solaris packaging system provides a way of installing pieces of the file system in logical boundaries, first by defining packages and a set of tools to manipulate them. Furthermore, the package definition, `pkginfo(4)`, defines the `SUNW_PKGTYPE` parameter which can be used to differentiate packages that belong in the root file system proper (which might be installed

in a non-global zone) and those that belong somewhere else such as `/usr` (which might only be installed in the global zone).

Finally, since various unbundled and third-party packages may have dependencies on Solaris packages that may or may not be installed in a non-global zone, either the packaging database in a non-global zone will need to have knowledge about the packages it is "importing" from the global zone or restrictions will need to be placed on the types of configurations that will be supported. This project proposes to do the latter in order to simplify the dependency issues.

12.2 Zone Models

There will be two supported configurations for a non-global zone delivered by the project. The first will provide the maximum configurability by installing all of the required and any selected optional Solaris packages into the zone. The second will optimize the sharing of objects by only installing a subset of the "root" packages (those with `SUNW_PKGTYPE` set to "root") and using read-only lofs file systems to gain access to other files. Both models can exist on the same system although converting a zone from one model to another will not be supported except through an explicit zone destruction and then reinstallation.

12.2.1 Whole Root Model

In this particular model, all packages required for a zone that make up a particular metacluster will be installed as well as any other packages selected by the global administrator. The advantages of this model include the ability for the zone administrator to customize their zone (for example, creating a `/usr/local`) and to add unbundled or third-party packages that might have dependencies on arbitrary Solaris packages (such as `SUNWcsu`).

The disadvantages of this model include the loss of sharing of text segments from executables and shared libraries by the virtual memory system and a much heavier disk footprint for each zone that is configured this way.

12.2.2 Sparse Root Model

In this model, only certain "root" packages will be installed in the non-global zone. This will include a subset of the required root packages that are normally installed in the global zone as well as any additional root packages that the global administrator might have selected. Access to other files will be via read-only lofs file systems and would include the directories `/lib` and `/usr` (other possibilities include the `/opt` and `/sbin` directories). This is similar to the way a diskless client is configured where `/usr` and other file systems are mounted over the network with NFS.

The advantages of this model are greater performance due to the efficient sharing of executables and shared libraries and a much smaller disk footprint for that zone itself.

The disadvantages are that the zone administrator will not be able to arbitrarily add packages or software that might have a dependency on packages that don't exist in the zone. In this case, the global administrator will need to add this software to the global zone and then allow that part of the file system space to be shared.

12.3 Package identification

There is currently no mechanism in the packaging infrastructure to identify which packages are suitable to be installed in non-global zones and which should only be installed in the global zone. Identifying which packages should go where is under investigation as part of the project. One potential solution would be to introduce a new packaging attribute, `SUNW_ZONETYPE`, which would appear in the `pkginfo(4)` file of a package. If this attribute is present and set to "global", then this package would only be installed in the global zone. If the attribute is set to "any", this package would be potentially available for a non-global zone if it was selected. For backwards compatibility, a missing attribute would be interpreted as meaning that this package is also a candidate for a non-global zone.

The Solaris Process Model Unification project (PSARC/2002/117) [14] moved many of the shared libraries that previously existed in `/usr/lib` into `/lib`. As a result, these libraries were moved from various "usr" packages into several "root" packages. Since the `/lib` directory is an important directory to be shared by Sparse Root model zones, there needs to be a way of identifying packages that deliver into this directory in the same way that the `SUNW_PKGTYPE=usr` attribute identifies objects under `/usr`. This project is looking into whether an additional value for this attribute (`SUNW_PKGTYPE=lib`) is required to identify such packages.

12.4 Package refactoring

Some packages which are typically installed in the root file system include kernel modules such as those delivered under `/kernel` and `/platform`. Although these files do no harm if installed in the root file system of a non-global zone, they do take up large amounts of disk space and ideally should not be installed if possible.

In addition, there are certain files under directories such as `/etc` which serve no purpose in a non-global zone and might be a source of confusion for the zone administrator. Some examples of these include the device databases (such as `/etc/minor_perm`) administered through `add_drv(1M)` and `update_drv(1M)`, `/etc/system`, configuration files for certain networking frameworks such as PPP, IPQoS and NCA which can only be configured from within the global zone and startup scripts under `/etc/init.d` which start processes which cannot run outside of the global zone.

Fortunately, most of the current Solaris packages are structured in such a way that installing them makes sense in either the global zone only or in both global and non-global

zones. There are a very small number of packages, however, which contain a mixture of objects that belong only in the global zone (mostly kernel modules) and objects that potentially could belong in non-global zones as well. This project proposes to "refactor" those packages so that the packaging system can more easily distinguish which packages are eligible for non-global zones and so that irrelevant files aren't needlessly installed in them.

12.5 Zone installation

When the `zoneadm(1M)` command is used to install a zone, it or some underlying program will prompt for certain options including which model should be used for the zone in question and which metacluster of packages to install. The actual interactive user interface is still under investigation as is whether Jumpstart can be used to provide a profile for a zone to be created.

There also needs to be a repository to pull down packages so that the zone's file system can be populated. This project proposes to require the use of distribution media like a DVD or a net-install image in order to obtain the necessary packages. The use of Flash archives to populate a zone's file system is also under investigation.

12.6 Patch and Release Dependencies

One complication with respect to maintaining software on a system with zones is the dependency between the user-level software installed in a zone and the kernel software running on the machine where the zone is configured. Since the kernel software is system-wide, the user-level software in each zone must be compatible with the kernel software installed in the global zone. This implies that patches that contain both user and kernel components, or any dependent patches, must be applied to all zones in the system.

Although it is technically possible for some system software (e.g., commands based on stable interfaces) to differ in version between zones, there is no existing infrastructure for systematically distinguishing between such software and that which is more tightly coupled with kernel changes (e.g., libthread). Thus, this project plans to require that (at a minimum) all software bundled in Solaris must be kept at an equivalent version in each zone; that is, any patch or upgrade must be applied across all zones in the system. Each zone may contain zone-specific customizations, and in fact the packages installed in each zone may differ, but any zone that contains a given package must have the same version of that package. This project plans to update the various install and patch tools to make it easy (perhaps automatic) to upgrade or patch all zones (including the global zone) simultaneously.

Chapter 13

Interoperability and Integration Issues

13.1 Project Dependencies

This project depends on a number of others. Some of these have already integrated into Solaris 10; others must integrate prior to the integration of this project. These dependencies are:

Least Privilege (PSARC/2002/188) This project provides fine-grained kernel privileges, which are used to enforce zone privilege restrictions. See Section 5.2 and the ARC documentation [7] for more details.

Read-only lofs (PSARC/2001/718) This project changes the `lofs` file system to prevent writes to files when mounted read-only. This is needed to prevent cross-zone communication and interference when multiple zones are sharing the same file system contents using `lofs`. See Chapter 7 for more information.

System V IPC Controls (PSARC/2002/694) This project removes a number of system-wide `/etc/system` tunables controlling System V IPC objects, and adds resource controls that can be used to restrict usage if necessary. This is needed since system-wide tunables are not generally useful in a system with zones. See Section 11.4 for more details.

libpool(3LIB) enhancements (PSARC/2003/136) This project extends the `libpool` interface and moves much of the infrastructure for managing resource pools into the kernel. This is needed to support binding zones to pools, and to provide a localized view of pool resources within a zone. See Section 10.1.3 for more information.

13.2 Other Related Solaris Projects

A number of other projects under development for Solaris either create opportunities to make the zone facility more capable or complete, or represent new ways in which zones might be

used. Although these are not considered dependencies for this case, they may result in some follow-on projects to provide better integration with zones.

The Solaris Fault Management Architecture (PSARC/2002/412) [32] will provide a protocol and set of APIs for monitoring hardware fault events; these mechanisms could potentially be used to restrict the actions taken when a fault is detected that affects only user-level process state. In essence, rather than rebooting the entire system, only the affected zone could be rebooted.

Greenline (PSARC/2002/547) [10] provides a management framework for long-running services in Solaris. This project will include separating the current tasks performed by `init(1M)` into more granular software components, and building a general framework for starting (and restarting) applications based on explicit dependencies. While such functionality isn't strictly needed to support zones, it does potentially provide an elegant mechanism for automating the startup of applications within a zone without dragging in the console management aspect of `init(1M)`.

ZFS (PSARC/2002/240) [4] is a new file system for Solaris that introduces a number of new features. Among these is the ability to easily and flexibly control the amount of storage consumed by a given file system. This would provide a way to implement per-zone "quotas" by restricting a zone to using one or more of these file systems, or even allowing the assignment of a pool of storage to a zone and allowing the zone administrator to allocate that storage to file systems as needed. Something similar can be done today with the soft partitioning technology available in SVM and VxVM (or even with `lofi(7D)` if performance is not critical), but ZFS should make this type of usage much more convenient and straightforward.

13.3 Trusted Solaris

While the Trusted Solaris Operating Environment is based on Solaris, it currently requires a large number of changes to Solaris, including extensive kernel changes. As such, it is built and delivered as a completely separate product, with a separate patch tree and hardware qualification matrix. This is very costly to maintain, and makes it difficult to expand the ISV and customer base for the product.

A project [11] is underway to improve this situation by implementing the Trusted Solaris functionality as a layer on top of Solaris. The use of zones is key to this effort; zones will be used to provide the isolation formerly provided through the use of labels implemented in the kernel.

13.4 Sun Cluster

The Sun Cluster product includes a Resource Group Manager (RGM) that provides a framework for managing highly available services in a clustered environment. There will likely be

interest in deploying such services in a zone environment, so that HA services can take advantage of the isolation and security provided by zones. The project team is currently discussing this possibility with members of the RGM development team. The current (tentative) plan is that the RGM infrastructure would run in the global zone, but be able to manage services in non-global zones. The details of how this will work are outside the scope of this case.

13.5 N1

The Sun Network 1 (N1) initiative is defining a "meta-architecture" for managing a network of computers as a single pool of resources, dynamically provisioned to meet the needs of network services [26]. In the N1 model, systems are virtualized so that their specific identity does not matter; as resource requirements change, applications can be freely moved between systems. Such an approach seems to naturally fit with the zone model, where applications are already isolated from details of the physical systems. In addition, zones would allow systems to be subdivided, so that multiple distinct applications or services could be deployed on the same physical system; this provides for increased granularity of control and further helps maximize utilization across the data center.

13.6 Third-Party Kernel Modules

Third-party kernel modules present a problem for zones because there is no intrinsic way to determine whether a given module obeys the zone security requirements. Hence, the ability to load a kernel module is restricted to the global zone, and modules in the standard module search paths cannot be modified from within a non-global zone. In addition, arbitrary device nodes cannot be created within a non-global zone, and the availability of specific devices in non-global zones is part of the zone configuration and must be specifically enabled by a global administrator. Note that these issues do not affect the ability to use third-party kernel modules within the global zone; such usage should continue to work as well as it did prior to the integration of zones, although the changes to the `cred_t` structure may mandate some code modification.

13.7 Third-Party Applications

A design goal is to avoid breaking third party applications, or to minimize such breakage when it must occur. The "big 4" open source applications for ISPs (sendmail, named, apache and ssh) are all known to work unmodified within a zone.

Applications that are installed into a zone and deliver start/stop scripts into `/etc/init.d` and one or more of the `/etc/rc?.d` directories will be started when the zone is booted. This may have unintended consequences; applications that should not be run within a zone

(perhaps because they depend on interfaces or services that are not available within a non-global zone) may be started automatically. Such applications should either not be installed into non-global zones, should have their start scripts modified so that they do not start in non-global zones, or should take advantage of the extended package attributes described in Chapter 12 to deliver different contents into global and non-global zones. The alternative of only starting applications automatically when specifically modified to start in a zone (e.g., by creating a separate `/etc/rc2.d` directory and run level for non-global zones) was seen as too burdensome for ISVs and customers. Fortunately, future integration with Greenline should help ameliorate these issues.

Bibliography

[1] Partitioning for the IBM eserver pSeries 690 system. http://taxman.eng/server-virtualization/Regatta_LPAR.pdf.

[2] System partitioning on IBM eserver xSeries servers. http://taxman.eng/server-virtualization/xseries.pdf.

[3] Hamid Asayesh. IPv6 protocol stack and utilities. PSARC/1997/184. http://sac.eng/PSARC/1997/184/commit.materials/tcpip.ps.

[4] Jeff Bonwick. ZFS (Zettabyte filesystem). PSARC/2002/240. http://sac.eng/PSARC/2002/240/incept.materials/onepager.txt.

[5] Eric Cheng and Jerry Chu. TCP_IOC_ABORT_CONN ioctl. PSARC/2001/292. http://sac.eng/PSARC/2001/292/mail.

[6] Chris Dalton and Tse Huong Choo. An operating system approach to securing e-services. *Communications of the ACM*, 44(2), February 2001. http://taxman.eng/server-virtualization/Dalton.ACM.pdf.

[7] Casper Dik. Least privilege for Solaris. PSARC/2002/188. http://sac.eng/PSARC/2002/188/final.materials/priv.pdf.

[8] Andrei Dorofeev and Andrew Tucker. Revised share scheduler. PSARC/2000/452. http://sac.eng/PSARC/2000/452/commit.materials/design-document.html.

[9] Ashley Saulsbury et al. Project Q: Sun X-PARs. http://projectq.eng/docs/q-spec-049c.pdf.

[10] Jonathan Adams et al. Greenline draft architecture: Alpha. PSARC/2002/547. http://sac.eng/PSARC/2002/547/incept.materials/alpha-design.pdf.

[11] Glenn Faden. Layered Trusted Solaris. PSARC/2002/762. http://sac.eng/Archives/Projects/2002/20021212_glenn.faden.

[12] Glenn Faden. Server virtualization with Trusted Solaris™8 Operating Environment. *Sun Blueprints™ Online*, February 2002. http://taxman.eng/server-virtualization/trustedsoe.pdf.

[13] Marty Faltesek. Devfsadm. PSARC/1997/202. `http://sac.eng/PSARC/1997/202/commit.materials/devfsadm.txt`.

[14] Roger Faulkner. Solaris process model unification. PSARC/2002/117. `http://sac.eng/PSARC/2002/117/commit.materials/motivation`.

[15] Jason Goldschmidt. 6to4 router. PSARC/2001/471. `http://sac.eng/PSARC/2001/471/commit.materials/FuncSpec.html`.

[16] Stephen Hahn. Task-based resource controls. PSARC/2000/137. `http://sac.eng/PSARC/2000/137/commit2.materials/Resource_controls.ps`.

[17] Stephen Hahn, Ozgur Leonard, and Gary Pennington. 'Resource Pools': Administrative support for processor sets and extensions. PSARC/2000/136. `http://sac.eng/PSARC/2000/136/commit.materials/design-document.ps`.

[18] Vikram Hegde. Libdevinfo devlinks interfaces. PSARC/2000/310. `http://sac.eng/PSARC/2000/310/mail`.

[19] R. Hinden and S. Deering. RFC 2373: IP version 6 addressing architecture, July 1998. `http://www.ietf.org/rfc/rfc2373.txt?number=2373`.

[20] Hans Hoffman. Sysevent extensions: General purpose event channels. PSARC/2002/321. `http://sac.eng/PSARC/2002/321/mail`.

[21] Poul-Henning Kamp and Robert Watson. Jails: Confining the omnipotent root. In *2nd International System Administration and Networking Conference (SANE 2000)*, May 2000. `http://taxman.eng/server-virtualization/jail.ps.gz`.

[22] Ozgur Leonard. Read-only lofs. PSARC/2001/718. `http://sac.eng/PSARC/2001/718/mail`.

[23] Michael Lim. Solaris IP filter. `http://sac.eng/Archives/Projects/2003/20030127_jon.krueger`.

[24] Tim Marsland. Jails in Solaris? In *SSG Technical Conference*, March 2001. `http://taxman.eng/internal-documents/suntech-2001/jails.ps`.

[25] Jean-Christophe Martin. Replace `db_lid` with `db_projid` in `dblk_t`. PSARC/2001/210. `http://sac.eng/PSARC/2001/210/specification`.

[26] N1 principles of operation. `http://new-webhome2.eng/Network1/n1-meta-poo.pdf`.

[27] Bao Phan. Solaris random number generator. PSARC/2000/484. `http://sac.eng/PSARC/2000/484/final.materials/whitepaper.ps`.

[28] Kacheong Poon. New `db_uid` and `db_lid` `dblk_t` fields. PSARC/1999/341. `http://sac.eng/PSARC/1999/341/specification`.

[29] David Powell. System V IPC resource controls. PSARC/2002/694. `http://sac.eng/PSARC/2002/694/mail`.

[30] Daniel Price. Generic SCSI pass through for Solaris. PSARC/1999/525. `http://sac.eng/PSARC/1999/525/mail`.

[31] Sebastien Roy. IPv6 default address selection. PSARC/2002/390. `http://sac.eng/PSARC/2002/390/incept.materials/functional-spec.html`.

[32] Michael Shapiro and Cynthia McGuire. Solaris fault management architecture. PSARC/2002/412. `http://sac.eng/PSARC/2002/412/incept.materials/fma-paper.pdf`.

[33] Andrew Tucker. Acacia design overview. `http://taxman.eng/internal-documents/acacia-design-overview.ps`.

6046955 8.050903

Sun Proprietary/Confidential: Internal Use Only

Appendix A

Interface Tables

Tables A.1 and A.2 show the interfaces introduced and modified by this project, respectively.

Interface Name	Classification	Comment
`zlogin(1)`	Evolving	Utility to enter a zone
`zonename(1)`	Evolving	Utility to get current zone name
`zoneadm(1M)`	Evolving	Utility to administer zones
`zonecfg(1M)`	Evolving	Utility to configure zones
`getzoneid(2)`	Evolving	System call to get current zone id
`zone_create(2)`	Evolving	System call to create a zone
`zone_destroy(2)`	Evolving	System call to destroy a zone
`zone_enter(2)`	Evolving	System call to enter a zone
`zone_list(2)`	Evolving	System call to list active zones
`ucred_getzoneid(3C)`	Evolving	Function to get zone id from credential

`getzonebynam(3C)`
`getzonebyid(3C)`
`getzoneent(3C)`
`setzoneent(3C)`
`endzoneent(3C)`
	Evolving	Functions to access zone information

`crgetzoneid(9F)`	Evolving	Kernel interface to access zone id

Table A.1: New Interfaces

Interface Name	Classification	Comment
`ipcrm(1)`	Standard	Added `-z` option
`ipcs(1)`	Standard	Added `-z` and `-Z` options
`pgrep(1)`	Evolving	Added `-z` option
`pkill(1)`	Evolving	Added `-z` option
`priocntl(1)`	Standard	Added `-i zoneid` option
`ps(1)`	Standard	Added `-o zone` and `-o zoneid` options
`renice(1)`	Standard	Added `-i zoneid` option
`ifconfig(1M)`	Stable	Added `-Z`, `zone`, and `-zone` options
`poolbind(1M)`	Evolving	Added `-i zoneid` option
`prstat(1M)`	Evolving	Added `-z` and `-Z` options
`mount_proc(1M)`	Evolving	Added `-o zone` option
`priocntl(2)`	Standard	Added `P_ZONEID` id type
`getpriority(3C)` `setpriority(3C)`	Standard	Added `PRIO_ZONE` to possible "which" values.
`proc(4)`	Stable	Added zone id to `pstatus_t` and `psinfo_t`
`if_tcp(7P)`	Stable	Added `SIOCGLIFZONE` and `SIOSLIFZONE` Added `LIFC_ALLZONES` for `SIOCGLIFCONF` and `SIOCGLIFNUM`

Table A.2: Modified Interfaces

Appendix B

Document Type Definition for zonecfg

The following is the DTD for zonecfg(1M). Note that it is Project Private, and is currently incomplete; that is, it does not specify definitions for all of the resource types and properties we expect to be part of a zone configuration when this project is complete. It is included for illustrative purposes only.

```
<?xml version='1.0' encoding='UTF-8' ?>

<!--
    Copyright 2002-2003 Sun Microsystems, Inc.  All rights reserved.
    Use is subject to license terms.

    ident       "@(#)zonecfg.dtd.1      1.4     03/03/28 SMI"
-->

<!--Element Definitions-->

<!ELEMENT filesystem    EMPTY>

<!ATTLIST filesystem    special         CDATA #REQUIRED
                        directory       CDATA #REQUIRED
                        type            CDATA #REQUIRED
                        options         CDATA "">

<!ELEMENT network       EMPTY>

<!ATTLIST network       virtual         CDATA #REQUIRED
                        address         CDATA #REQUIRED
                        physical        CDATA #REQUIRED>

<!ELEMENT device        EMPTY>
```

```
<!ATTLIST device        name            CDATA #REQUIRED
                        matchtype       (drv_min | driver | devpath )
                                        #REQUIRED
                        match           CDATA #REQUIRED>

<!ELEMENT zone          (filesystem | network | device)*>

<!ATTLIST zone          name            CDATA #REQUIRED
                        rootpath        CDATA #REQUIRED
                        version         NMTOKEN #FIXED '1'>
```

Appendix C

Man Pages for New Interfaces

C.1 User Commands

C.1.1 zlogin(1)

```
NAME
     zlogin - Enter a zone

SYNOPSIS
     zlogin [ -CE ] [ -e c ] [ -l username ] zonename

     zlogin [ -E ] [ -e c ] [ -l username ]  zonename  utility  [
     argument ... ]

DESCRIPTION
     The zlogin utility is used by the administrator to enter  an
     operating  system  zone.  Only  a superuser operating in the
     global system zone may use this utility.

     zlogin operates in one of two modes.  If no utility argument
     is  given  and the stdin file descriptor for the zlogin pro-
     cess is a tty device, zlogin operates in  interactive  mode.
     In  this  mode,  it  creates  a  new pseudo terminal for use
     within the login session; programs requiring  a  tty  device
     (for  example, vi(1)) will work properly in this mode.  Oth-
     erwise, zlogin operates in non-interactive mode;  this  mode
     may  be  useful  for  script authors since stdin, stdout and
     stderr are preserved and  the  exit  status  of  utility  is
     returned upon termination.

     Except when the  -C  option  is  specified,  zlogin  invokes
```

su(1M) once login to the zone has occurred in order to set up the users environment. In interactive mode, the "-" argument is passed to su in order to provide a login session.

If the -C option is specified, the user is connected to the zone console device. The zone console persists across zone reboot; if the zone is halted, the console will disconnect.

OPTIONS

The following options are supported:

-C Connect to the zone console.

-E Disable the ability to access extended functions or to disconnect from the login via the escape sequence character.

-e c Specify a different escape character, c, for the key sequence used to access extended functions and to disconnect from the login.

-l Specify a different username for the zone login. If you do not use this option, the zone username used is "root". This option is invalid if the -C option is specified.

SECURITY

Once a process has been placed in a zone other than zone 0, the process cannot change zone again, nor can any of its children.

OPERANDS

zonename

The name of the zone to be entered.

utility

The utility to be run in the specified zone.

argument...

Arguments passed to the utility.

EXIT STATUS

The zlogin utility exits with one of the following values:

```
     0    Success.

     1    Permission denied, or failure to enter the zone.

     Any  Return code from utility, if operating in non-
          interactive mode.

ATTRIBUTES
     See attributes(5) for descriptions of the following attri-
     butes:

     ----------------------------------------------------------
    |       ATTRIBUTE TYPE        |       ATTRIBUTE VALUE      |
    |-----------------------------|-----------------------------|
    | Availability                | SUNWcsu                     |
    |-----------------------------|-----------------------------|
    | Interface Stability         | Evolving                    |
    |-----------------------------|-----------------------------|

SEE ALSO
     su(1M), zoneadm(1M), zonecfg(1M), zone_enter(2)
```

C.1.2 zonename(1)

```
NAME
     zonename - print name of current zone

SYNOPSIS
     zonename

DESCRIPTION
     The zonename command prints the name of the current zone.

ATTRIBUTES
     See attributes(5) for descriptions of the following attri-
     butes:

     ----------------------------------------------------------
    |       ATTRIBUTE TYPE        |       ATTRIBUTE VALUE      |
    |-----------------------------|-----------------------------|
    | Availability                | SUNWcsu                     |
    |-----------------------------|-----------------------------|
    | Interface Stability         | Evolving                    |
```

```
|_____________________________|_____________________________|

SEE ALSO
     zlogin(1), zoneadm(1M), zonecfg(1M).
```

C.2 System Administration Commands

C.2.1 mount_proc(1M)[1]

```
NAME
     mount_proc - mount proc file systems

SYNOPSIS
     mount [ -F proc ] [ -o zone=zoneid ] [ -O ] special
          mount_point

DESCRIPTION
     proc is a memory based file system which provides access  to
     the  state of each process and light-weight process (lwp) in
     the system.

     mount attaches a proc file system to the file system hierar-
     chy at the pathname location mount_point, which must already
     exist.  If mount_point has any contents prior to  the  mount
     operation, these remain hidden until the file system is once
     again unmounted.  The attributes (mode, owner, and group) of
     the  root  of  the  proc  filesystem  are inherited from the
     underlying mount_point, provided that those  attributes  are
     determinable.  If  not,  the  root's  attributes are set to
     their default values.

     The special argument is usually specified as /proc but is in
     fact disregarded.

OPTIONS
     -o zone=zoneid The zoneid argument specifies that  the  proc
                    file  system  is to be provided to the speci-
                    fied zone, in which case it will contain only
                    those items within the zone, and for conveni-
                    ence, read-only access to  process  0  and  1
                    also.
```

[1]The mount_proc(1M) command is not new with this project, but it did not previously have a man page.

```
     -O             Overlay mount.  Allow the file system  to  be
                    mounted  over an existing mount point, making
                    the underlying file system inaccessible.   If
                    a  mount is attempted on a pre-existing mount
                    point without setting this  flag,  the  mount
                    will fail, producing the error "device busy".

FILES
     /etc/mnttab          table of mounted file systems

ATTRIBUTES
     See attributes(5) for descriptions of the  following  attri-
     butes:
     ____________________________________________________________
    |       ATTRIBUTE TYPE        |       ATTRIBUTE VALUE        |
    |_____________________________|_____________________________|
    | Availability                | SUNWcsu                      |
    |_____________________________|_____________________________|
    | Interface Stability         | Evolving                     |
    |_____________________________|_____________________________|

SEE ALSO
     mount(1M),  zoneadm(1M),  zonecfg(1M),  mkdir(2),  mount(2),
     open(2), umount(2), zone_create(2), mnttab(4), attributes(5)
     proc(4)

NOTES
     If the directory on which a file system is to be mounted  is
     a symbolic link, the file system is mounted on the directory
     to which the symbolic link refers, rather than on top of the
     symbolic link itself.
```

C.2.2 zoneadm(1M)

```
NAME
     zoneadm - administer zones

SYNOPSIS
     zoneadm help

     zoneadm -z zonename [-v] subcommand [subcommand options]
```

```
zoneadm -Z [-v] subcommand [subcommand options]

DESCRIPTION
     The zoneadm utility is used to administer system zones.  A
     zone is an application container maintained by the operating
     system runtime.

SECURITY
     Once a process has been placed in a zone other than zone 0,
     the process cannot change zone again, nor can any of its
     children.

SUBCOMMANDS
     The following subcommands are supported.

     help Print usage message.

     boot [-n]
          Activates (boots) specified zone(s).  The -n option
          starts the zone without running init scripts.  Prior to
          activation, any previous runtime resources of the
          zone(s) (mounts, System V IPC objects, etc.) are
          removed.  This operation fails if the zone(s) is (are)
          already running.

     shutdown [-f]
          Shuts down specified zone(s).  The difference between
          halt and shutdown is that shutdown results in all shut-
          down scripts being executed in an orderly fashion -
          this is equivalent to running 'init 0' inside the
          zone(s) - whereas halt bypasses running the shutdown
          scripts.  These operations also remove runtime
          resources of the zone(s).  The -f option can be used to
          force things, in case something gets stuck.

     halt [-f]
          Halts specified zone(s).  The difference between halt
          and shutdown is that shutdown results in all shutdown
          scripts being executed in an orderly fashion - this is
          equivalent to running 'init 0' inside the zone(s) -
          whereas halt bypasses running the shutdown scripts.
          These operations also remove runtime resources of the
          zone(s).  The -f option can be used to force things, in
          case something gets stuck.
```

```
     reboot [-nf]
          Restarts the zone(s) (equivalent to a shutdown / boot
          sequence).  Fails if the zone(s) is (are) not active.
          The -f option can be used to force things, in case
          something gets stuck while shutting things down. The
          -n option results in the zone(s) rebooting without run-
          ning init scripts.

     info [-n]
          Displays the name of the specified zone(s).  The gen-
          eral -v option can be used to display additional infor-
          mation: the zone name, id, current state, and root
          directory.  The -n option will print the zone's numeric
          id instead of its name; in verbose mode, -n will cause
          the name not to be printed:  "??" will be printed
          instead.

     verify
          Check to make sure the configuration of the specified
          zone(s) can safely be installed on the machine: physi-
          cal network interfaces exist, rootpath and its parent
          directory exist and are owned by root with appropriate
          modes (755 for rootpath), etc.

     install
          Install the configuration of the specified zone(s) on
          to the system.  Note that this will automatically
          attempt to verify first, and refuse to install if the
          verify step fails.

     destroy
          Uninstall the configuration of the specified zone(s)
          from the system.

OPTIONS
     -z   zonename String identifier for a zone.

     -Z   Indicates all zones.

     -v   Requests verbose output.

EXIT STATUS
     The zoneadm command exits with one of the following values:
```

```
     0    Success.

     1    Failure.

ATTRIBUTES
     See attributes(5) for descriptions of the  following  attri-
     butes:
     ---------------------------------------------------------
     |       ATTRIBUTE TYPE        |       ATTRIBUTE VALUE      |
     |-----------------------------|-----------------------------|
     | Availability                | SUNWcsu                     |
     |-----------------------------|-----------------------------|
     | Interface Stability         | Evolving                    |
     |-----------------------------|-----------------------------|

SEE ALSO
     zlogin(1),   zonecfg(1M),    zonename(1),    zone_create(2),
     zone_list(2).
```

C.2.3 zonecfg(1M)

```
NAME
     zonecfg - Set up zone configuration

SYNOPSIS
     zonecfg -z zonename

     zonecfg -z zonename subcommand

     zonecfg -z zonename -f command-file

DESCRIPTION
     The first format of the zonecfg command is  for  interactive
     usage.  The zonecfg command is used to create and modify the
     configuration for a zone.

SUBCOMMANDS
     The following subcommands are supported:

     help [usage] [commands] [syntax] [<command-name>]
          Prints usage message.

     create
```

```
     Creates a "blank" configuration for the specified zone.

import template
     Creates a configuration identical to the specified tem-
     plate, but with template changed to zone.

export
     Prints configuration to stdout in a form  suitable  for
     use in a command-file.

add resource-type resource-id
     Add specified resource to configuration.

destroy
     Destroys the specified zone.

remove resource-type resource-id
     Remove specified resource from configuration.

setprop resource-type resource-id property-type property-id
     Sets property values within a resource.

unsetprop resource-type resource-id property-
     type [property-id]
     Unsets properties within a resource.

info [resource-type [resource-id]]
     Displays information about the  current  configuration.
     If  resource-type  is specified, displays only informa-
     tion  about  resources  of  the  relevant   type.    If
     resource-id  is  specified,  displays  only information
     about that resource.

verify
     Verifies current configuration  for  correctness  (some
     resource types have required properties).

commit
     Commits current configuration.  Configuration  must  be
     committed  to be used by zoneadm.  Until the configura-
     tion is committed, changes  can  be  removed  with  the
     reset subcommand.  This operation is attempted automat-
     ically upon completion of a zonecfg session.
```

```
     revert
          Reverts configuration back to the last committed state.

RESOURCES
     For resource type of:

     rootpath     resource-id is path to zone root

     fs           resource-id is mount point path

     net          resource-id is virtual interface name

     device       resource-id is a device matching specifier

PROPERTIES
     For resource type ... there are property types ...:

     rootpath     none

     fs           special, type, options

     net          address, physical

     device       matchtype, match

OPERANDS
     zonename
          The name of a zone.  Zone  names  are  case-sensitive;
          they  must begin with an alpha-numeric, and can contain
          alpha-numerics plus the _ and - characters.  The  names
          blank,  default  and  help  are reserved and may not be
          used.
EXIT STATUS
     The zonecfg command exits with one of the following values:

     0    Success.
     1    Operation failed.

EXAMPLES
     In the following example, zonecfg is being  used  to  create
     the  environment  for  a new zone with three logical network
     interfaces.

          example# zonecfg -z my-zone3
```

```
          zonecfg> import default
          zonecfg> add net eri01
          zonecfg> add rootpath /export/home/my-zone3
          zonecfg> setprop net eri01 address 192.168.0.1
          zonecfg> setprop net eri01 physical eri0
          zonecfg> add net eri02
          zonecfg> setprop net eri02 address 192.168.0.2
          zonecfg> setprop net eri02 physical eri0
          zonecfg> add net eri03
          zonecfg> setprop net eri03 address 192.168.0.3
          zonecfg> setprop net eri03 physical eri0
          example#

ATTRIBUTES
     See attributes(5) for descriptions of the  following  attri-
     butes:

     ____________________________________________________________
    |       ATTRIBUTE TYPE        |       ATTRIBUTE VALUE       |
    |_____________________________|_____________________________|
    | Availability                | SUNWcsu                     |
    |_____________________________|_____________________________|
    | Interface Stability         | Evolving                    |
    |_____________________________|_____________________________|

SEE ALSO
     zlogin(1M), zoneadm(1M).
```

C.3 System Calls

C.3.1 getzoneid(2)

```
NAME
     getzoneid - get zone ID

SYNOPSIS
     #include <zone.h>

     zoneid_t getzoneid(void);

DESCRIPTION
     The getzoneid() function returns the zone ID of the  calling
     process.
```

```
RETURN VALUES
     See DESCRIPTION.

ERRORS
     The function can't fail.

ATTRIBUTES
     See attributes(5) for descriptions of the  following  attri-
     butes:
```

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	Evolving
MT-Level	Safe

```
SEE ALSO
     Intro(2), zone_enter(2), zone_create(2), chroot(2).
```

C.3.2 zone_create(2)

```
NAME
     zone_create, zone_destroy - create and destroy zone

SYNOPSIS
     #include <zone.h>

     int zone_create(zoneid_t zoneid, const char *rootpath,
          const char *nodename,priv_set_t* privs );

     int zone_destroy(zoneid_t zoneid);

DESCRIPTION
     These functions are used to create and destroy zones in  the
     kernel.

     The zone_create() function creates a new zone, with  a  root
     directory at rootpath, an initial node name of nodename, and
```

a privilege set of privs.

The rootpath parameter represents the root of the file system tree to which processes running in the zone will be restricted. This will be the starting point for path searches that begin with / (slash). In addition, the path names reported by mnttab(4) will be relative to the root directory for the zone when read from within the zone.

The nodename parameter is used as an initial value for the node name returned by hostname(1) and uname(1). It can be modified by the zone administrator by creating a nodename(4) file. This is distinct from the zone name reported by zonename(1), though the two may be the same in some cases.

The privs parameter is used to restrict the set of privileges available within the zone; any processes entering the zone will have their current privileges masked with those specified by privs. A zone must be created in the kernel using zone_create() before it is possible to enter the zone using zone_enter(2).

The zone_destroy() function removes the zone specified by the parameter zoneid from the kernel. The operation fails is there are any processes in the zone.

RETURN VALUES

Upon successful completion, these functions return 0. Otherwise, -1 is returned and errno is set to indicate the error.

ERRORS

The zone_create() function will fail if:

EACCES

Search permission is denied for a component of the path prefix of rootpath, or search permission is denied for the directory referred to by dirname.

EEXIST

A zone with an ID of zoneid is already active.

EFAULT

The location pointed to by rootpath does not exist.

EINVAL
The specified zoneid is invalid.

ELOOP Too many symbolic links were encountered in translating rootpath.

ENAMETOOLONG
The length of the rootpath argument exceeds PATH_MAX, or the length of a rootpath component exceeds NAME_MAX while _POSIX_NO_TRUNC is in effect.

ENOENT
The named directory does not exist or is a null pathname.

ENOLINK
The rootpath argument points to a remote machine and the link to that machine is no longer active.

ENOTDIR
Any component of the path name is not a directory.

EPERM The {PRIV_SYS_CONFIG} privilege is not asserted in the effective set of the calling process.

The zone_destroy() function will fail if:

EBUSY The zone could not be destroyed because processes exist within it.

EINVAL
The specified zoneid is invalid or does not exist.

EPERM The {PRIV_SYS_CONFIG} privilege is not asserted in the effective set of the calling process.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

Interface Stability	Evolving
MT-Level	Safe

```
SEE ALSO
     Intro(2),    zone_enter(2),     getzoneid(2),     chroot(2),
     privileges(5).
```

C.3.3 zone_enter(2)

```
NAME
     zone_enter - enter a zone

SYNOPSIS
     #include <zone.h>

     int zone_enter(zoneid_t zoneid);

DESCRIPTION
     The zone_enter() function  causes  the  calling  process  to
     enter  the  zone specified by the parameter zoneid.  On suc-
     cess, both the root directory and current working  directory
     of  the calling process will be set to the root directory of
     the zone.  A subsequent call to getcwd(3C) will return "/".

     In addition, all of the  calling  process's  privilege  sets
     will  be set to the intersection of their current value with
     the set specified as an argument to zone_create(2) when  the
     zone was created.

RETURN VALUES
     Upon successful completion, a value of 0 is returned.   Oth-
     erwise,  a value of -1 is returned and errno is set to indi-
     cate the error.

ERRORS
     zone_enter() will fail and the root  directory  will  remain
     unchanged if one or more of the following are true:

     EPERM                The {PRIV_SYS_CONFIG} privilege  is  not
                          asserted  in  the  effective  set of the
```

```
                    calling process, or the calling
                    process's permitted set is not a super-
                    set of the set specified to
                    zone_create(2).

     EINVAL         The zone has not been created in the
                    kernel with zone_create(2).

     EINVAL         The calling process is not in the global
                    zone.

ATTRIBUTES
     See attributes(5) for descriptions of the following attri-
     butes:
     ---------------------------------------------------------
     |       ATTRIBUTE TYPE        |       ATTRIBUTE VALUE      |
     |-----------------------------|----------------------------|
     | Availability                | SUNWcsu                    |
     |-----------------------------|----------------------------|
     | Interface Stability         | Evolving                   |
     |-----------------------------|----------------------------|
     | MT-Level                    | Safe                       |
     |-----------------------------|----------------------------|

SEE ALSO
     Intro(2), getzoneid(2), zone_create(2), chroot(2),
     getcwd(3C).
```

C.3.4 zone_list(2)

```
NAME
     zone_list - list zones

SYNOPSIS
     #include <zone.h>

     int zone_list(zoneid_t *zonelist, uint_t *numzones);

DESCRIPTION
     The zone_list() function returns a list of zones currently
     active in the system. On successful completion the number
     of zones active in the system will be stored in the location
     pointed to by numzones.
```

If zonelist is non-null, then zonelist points to a buffer where a list zones active in the system is to be stored, and numzones points to the maximum number of zone IDs the buffer can hold. On successful completion, the list of zones up to the maximum buffer size is stored in the buffer pointed to by zonelist.

RETURN VALUES

Upon successful completion, 0 is returned. Otherwise, -1 is returned and errno is set to indicate the error.

ERRORS

The zone_list() function will fail if:

EFAULT

The location pointed to by zonelist was not null and not writable by the user, or the location pointed to by numzones was not writable by the user.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	Evolving
MT-Level	Safe

SEE ALSO

Intro(2), zone_create(2), zone_enter(2), zone_getattr(2), getzoneid(2).

C.4 Library Interfaces

C.4.1 getzonebynam(3C)

```
NAME
     getzonebynam, getzonebyid, getzoneent, setzoneent, endzo-
     neent - get zone entry

SYNOPSIS
     #include <zone.h>

     struct zoneent *getzonebynam(const char *name);

     struct zoneent *getzonebyid(zoneid_t zoneid);

     struct zoneent *getzoneent(void);

     void setzoneent(void);

     void endzoneent(void);

DESCRIPTION
     These functions are used to obtain zone entries.

     The getzonebynam() function searches for a zone entry with
     the zone name specified by the character string parameter
     name.

     The getzonebyid() function searches for a zone entry with
     the (numeric) zone ID specified by the parameter zoneid.

     The setzoneent(), getzoneent(), and endzoneent() functions
     are used to enumerate zone entries from the database. set-
     zoneent() sets (or resets) the enumeration to the beginning
     of the set of zone entries. This function should be called
     before the first call to getzoneent(). Calls to getzo-
     nenam() and getzoneuid() leave the enumeration position in
     an indeterminate state. Successive calls to getzoneent()
     return either successive entries or NULL, indicating the end
     of the enumeration.

     The endzoneent() function may be called to indicate that the
     caller expects to do no further zone retrieval operations;
     the system may then close the zone file, deallocate
```

resources it was using, and so forth. It is still allowed, but possibly less efficient, for the process to call more zone functions after calling endzoneent().

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. The setzoneent() function may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to getzoneent(), the threads will enumerate disjoint subsets of the zone database.

RETURN VALUES

Zone entries are represented by the struct zoneent structure defined in <zone.h>:

```
struct zoneent {
    char *zone_name;     /* zone name */
    zoneid_t zone_id;    /* zone id */
    char *zone_home;     /* zone root directory */
    char *zone_comment;  /* zone descriptive comment */
    char *zone_attr;     /* zone attributes string */
};
```

The getzonebynam() and getzonebyid() functions each return a pointer to a struct zoneent if they successfully locate the requested entry; otherwise they return NULL.

The getzoneent() function returns a pointer to a struct zoneent if it successfully enumerates an entry; otherwise it returns NULL, indicating the end of the enumeration.

The getzonebynam(), getzonebyid(), and getzoneent() functions use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

Interface Stability	Evolving
MT-Level	Unsafe

```
SEE ALSO
     Intro(2), Intro(3), attributes(5), standards(5), zones(5)

NOTES
     Use of the enumeration interface getzoneent() is
     discouraged; enumeration is supported for the zone file, but
     in general is not efficient and may not be supported for all
     database sources.
```

C.5 Standards, Environments, and Macros

C.5.1 zones(5)

```
NAME
     zones - Solaris application containers

DESCRIPTION
     Note: this man page is obviously incomplete, but gives some
     idea of the kind of information that will be found here.

     The zone facility in Solaris provides an isolated environ-
     ment for running applications. Processes running in a zone
     are prevented from monitoring or interfering with other
     activity in the system. Access to other processes, network
     interfaces, file systems, devices, and inter-process commun-
     ication facilities are restricted to prevent interaction
     between processes in different zones. The privileges avail-
     able within a zone (see privileges(5)) are restricted to
     prevent operations with system-wide impact.

     Zones are configured and administered using the zonecfg(1M)
     and zoneadm(1M) utilities. These utilites let an adminis-
     trator specify the configuration details a zone, install
     file system contents (including software packages) into the
     zone, and manage the runtime state of the zone. The zlo-
     gin(1) utility allows an administrator to run commands
     within an active zone, without logging in through a
     network-based login server such as in.rlogind(1M) or
```

```
     sshd(1M).

     Each zone is identified by an alphanumeric name and a
     numeric ID. Both are configured using the zonecfg(1M) util-
     ity. The zonename(1) utility reports the current zone name.

     There is one special zone, the global zone, that refers to
     the standard operating environment in which a sufficiently
     privileged user has control over all aspects of the system.
     There is only one global zone, and it is automatically
     created when the system boots; if additional zones are not
     configured, all processes run in the global zone.

  Access Restrictions
     Processes running inside a zone (aside from the global zone)
     have restricted access to other processes. Only processes
     in the same zone will be visible through /proc (see proc(4))
     or through system call interfaces that take process IDs such
     as kill(2) and priocntl(2). Attempts to access processes
     that exist in other zones (including the global zone) will
     fail with the same error code that would be issued if the
     specified process did not exist. The exception to this rule
     is the processes with process IDs 0 and 1, which are visible
     in every zone but cannot be signalled or controlled in any
     way.
  Privilege Restrictions
  Global Zone Restrictions
SEE ALSO
     zlogin(1), zonename(1), in.rlogind(1M), sshd(1M),
     zoneadm(1M), zonecfg(1M), getzoneid(2), kill(2),
     priocntl(2), zone_create(2), zone_enter(2), zone_list(2),
     ucred_getzoneid(3C), getzonebynam(3C), proc(4),
     privileges(5), crgetzoneid(9F)
```

60469558.050903

Sun Proprietary/Confidential: Internal Use Only

Appendix D

Man Pages for Modified Interfaces

D.1 User Commands

D.1.1 ipcrm(1)

```
--- ipcrm.1.ref         Tue Apr  8 16:40:34 2003
+++ ipcrm.1.new         Tue Apr  8 16:40:34 2003
@@ -3,14 +3,23 @@
      memory ID

 SYNOPSIS
-     ipcrm  [-m shmid]  [-q msqid]  [-s semid]   [-M shmkey]   [-
-     Q msgkey] [-S semkey]
+     ipcrm [-z zone] [-m shmid] [-q msqid] [-s semid] [-M shmkey]
+     [-Q msgkey] [-S semkey]

 DESCRIPTION
      ipcrm removes one or more messages,  semaphores,  or  shared
      memory identifiers.

 OPTIONS
+     The following option is supported:
+
+     -z zone
+          Keys specified by options -M,  -Q,  and  -S  refer  to
+          facilities  in the specified zone (see zones(5)).  The
+          default is the zone in which the command is executing.
+          This  option  is  only useful when the command is exe-
+          cuted in the global zone.
+
      The identifiers are specified by the following options:
```

```
      -m shmid
@@ -68,4 +78,4 @@
 SEE ALSO
      ipcs(1),   msgctl(2),   msgget(2),   msgrcv(2),   msgsnd(2),
      semctl(2),   semget(2),   semop(2),   shmctl(2),   shmget(2),
-     shmop(2), attributes(5), environ(5), standards(5)
+     shmop(2), attributes(5), environ(5), standards(5), zones(5)
```

D.1.2 ipcs(1)

```
--- ipcs.1.ref        Tue Apr  8 16:40:34 2003
+++ ipcs.1.new        Tue Apr  8 16:40:34 2003
@@ -2,7 +2,7 @@
      ipcs - report inter-process communication facilities status

 SYNOPSIS
-     ipcs [-aAbcimopqst] [-D mtype]
+     ipcs [-aAbcimopqstZ] [-D mtype] [-z zone]

 DESCRIPTION
      The ipcs utility  prints  information  about  active  inter-
@@ -70,6 +70,19 @@
           shmdt(2 ) on shared memory  (see  shmop(2)),  time  of
           last semop(2) on semaphores. See below.

+     -z zone
+          Prints information about  facilities  associated  with
+          the  specified  zone  (see zones(5)).  The zone can be
+          specified as either a  name  or  a  numeric  id.   The
+          default  is  to  display information about the zone in
+          which the command is executing.  Note that this option
+          is only useful when executing in the global zone.
+
+     -Z   When executing in the global zone, prints  information
+          about  all zones.  Otherwise, prints information about
+          the zone in which the command is executing.  The  out-
+          put includes the zone associated with each facility.
+
      The column headings and the meaning of  the  columns  in  an
      ipcs  listing  are  given  below. The letters in parentheses
      indicate the options that cause the corresponding heading to
@@ -220,6 +231,9 @@
```

```
          The time the last semaphore operation was completed on
          the set associated with the semaphore entry.

+     ZONE        (Z)
+          The zone with which the facility is associated.
+
 ENVIRONMENT VARIABLES
      See environ(5) for descriptions of the following environment
      variables  that affect the execution of  ipcs: LANG, LC_ALL,
@@ -250,8 +263,9 @@
 SEE ALSO
      ipcrm(1),  msgget(2),  msgids(2),   msgrcv(2),   msgsnap(2),
      msgsnd(2),   semget(2),   semids(2),   semop(2),  shmctl(2),
-     shmget(2), shmids(2), shmop(2), attributes(5), environ(  5),
-     standards(5)
+     shmget(2), shmids(2), shmop(2), attributes(5), environ(5),
+     standards(5), zones(5)
+
 NOTES
      Things can change while ipcs is running. The information  it
      gives  is  guaranteed  to  be  accurate  only  when  it  was
```

D.1.3 pgrep(1)

```
--- pgrep.1.ref         Mon Apr  7 10:10:42 2003
+++ pgrep.1.new         Mon Apr  7 10:10:42 2003
@@ -5,13 +5,13 @@
 SYNOPSIS
      pgrep  [-flvx]  [-n  |  -o]   [-d delim]  [-P ppidlist]   [-
      g pgrplist]   [-s sidlist]   [-u euidlist]  [-U uidlist]  [-
-     G gidlist]  [-J projidlist]  [-t termlist]   [-T taskidlist]
-     [pattern]
+     G gidlist] [-J projidlist] [-t termlist] [-T taskidlist]  [-
+     z zoneid] [pattern]

      pkill  [-signal]  [-fvx]  [-n  |  -o]   [-P ppidlist]   [-
      g pgrplist]   [-s sidlist]   [-u euidlist]  [-U uidlist]  [-
-     G gidlist]  [-J projidlist]  [-t termlist]   [-T taskidlist]
-     [pattern]
+     G gidlist] [-J projidlist] [-t termlist] [-T taskidlist]  [-
+     z zoneid] [pattern]

 DESCRIPTION
```

```
     The pgrep utility examines the active processes on the  sys-
@@ -133,6 +133,11 @@
           cess argument string or executable file name match the
           pattern.

+     -z zoneidlist
+           Matches only processes whose zone ID is in  the  given
+           list.  Each  zone ID may be specified as either a zone
+           name or a numerical zone ID.
+
      -signal
           Specifies the signal to send to each matched  process.
           If no signal is specified, SIGTERM is sent by default.
```

D.1.4 priocntl(1)

```
--- priocntl.1.ref        Mon Apr  7 10:10:43 2003
+++ priocntl.1.new        Mon Apr  7 10:10:43 2003
@@ -40,10 +40,11 @@
     If an idlist is present, it must appear last on the  command
     line and the elements of the list must be separated by white
     space. If no idlist is present, an idtype argument  of  pid,
-    ppid,  pgid,  sid, taskid, class, uid, gid, or projid speci-
-    fies the process ID, parent process ID,  process  group  ID,
-    session  ID,  task  ID, class, user ID, group ID, or project
-    ID, respectively, of the priocntl command itself.
+    ppid,  pgid,  sid, taskid, class, uid, gid, projid, or zoneid
+    specifies the process ID, parent process ID,  process  group
+    ID,  session  ID, task ID, class, user ID, group ID, project
+    ID, or  zone  ID,  respectively,  of  the  priocntl  command
+    itself.

     The command

@@ -201,6 +202,11 @@
                command  applies to all processes with an effec-
                tive project ID equal to an ID from the list.

+          -i zoneid
+                idlist is a list of zone IDs. The priocntl  com-
+                mand  applies to all processes with an effective
+                zone ID equal to an ID from the list.
+
```

```
          -i all
               The priocntl command  applies  to  all  existing
               processes. No idlist should be specified (if one
```

D.1.5 ps(1)

```
--- ps.1.ref        Mon Apr  7 10:10:43 2003
+++ ps.1.new        Mon Apr  7 10:10:43 2003
@@ -365,6 +365,13 @@
            that  value  can  be  obtained; otherwise as a decimal
            integer.

+     zoneid
+           The zone  ID  number  of  the  process  as  a  decimal
+           integer.
+
+     zone  The zone ID of the process as a textual value if  that
+           value can be obtained; otherwise as a decimal integer.
+
      sid   The process ID of the session leader.

      taskid
```

D.1.6 renice(1)

```
--- renice.1.ref        Mon Apr  7 10:10:43 2003
+++ renice.1.new        Mon Apr  7 10:10:44 2003
@@ -63,8 +63,8 @@
            specifies  a  class  of  processes to which the renice
            command is to apply. The interpretation of the ID list
            depends on the value of idtype. The valid idtype argu-
-           ments are: pid, pgid, uid, gid, sid, taskid, and  pro-
-           jid.
+           ments are: pid, pgid, uid, gid, sid,  taskid,  projid,
+           and zoneid.

      -n increment
            Specifies how the system scheduling  priority  of  the
```

D.2 System Administration Commands

D.2.1 ifconfig(1M)

```
--- ifconfig.1m.ref        Tue Apr  8 16:40:35 2003
+++ ifconfig.1m.new        Tue Apr  8 16:40:35 2003
@@ -18,7 +18,7 @@
      tunnel_dest_address] [ token     address/prefix_length] [
      tsrc   tunnel_src_address] [trailers | -trailers]    [up]
      [down]  [xmit | -xmit]  [encaplimit  n | -encaplimit]  [tho-
-     plimit  n]
+     plimit  n] [zone zoneid | zone]

      /usr/sbin/ifconfig  interface  [address_family]  [  address
      [/prefix_length]     [dest_address]]   [   addif    address
@@ -494,6 +494,12 @@
      -xmit Disable transmission of packets on an  interface.  The
            interface will continue to receive packets.

+     zone zoneid
+           Place the IP interface in zone zoneid.
+
+     -zone Place  IP  interface  in  global  zone.  This  is  the
+           default.
+
 OPERANDS
      The interface operand, as well as  address  parameters  that
      affect it, are described below.
@@ -517,17 +523,20 @@
                that match.

                -a    Apply the commands to all interfaces in the
-                     system.
+                     user's zone.

                -d    Apply the commands to all "down" interfaces
-                     in the system.
+                     in the user's zone.

                -D    Apply the commands to all  interfaces   not
                      under DHCP (Dynamic Host Configuration Pro-
                      tocol) control.

                -u    Apply the commands to all  "up"  interfaces
```

```
-                    in the system.
+                    in the user's zone.
+
+               -Z    Apply the commands to all interfaces in the
+                     system.

               -4    Apply the commands to all IPv4 interfaces.

@@ -543,8 +552,8 @@
       address
               For the IPv4 family (inet), the address is either
               a  host  name  present in the host name data base
-              (see hosts(4)) or in the Network Information Ser-
-              vice   (NIS)   map  hosts,  or  an  IPv4  address
+              (see hosts(4))  or  in  the  Network  Information
+              Service  (NIS)  map  hosts,  or  an  IPv4 address
               expressed in the Internet standard "dot nota-
               tion".

@@ -708,10 +716,10 @@

     OFFLINE
           Indicates that the interface has  been  offlined.  New
-          addresses   cannot   be  created  on  this  interface.
-          Interfaces in an IP  network  multipathing  group  are
-          offlined   prior  to  removal  and  replacement  using
-          dynamic reconfiguration.
+          addresses  cannot be created on this interface. Inter-
+          faces in an IP network multipathing group are offlined
+          prior  to removal and replacement using dynamic recon-
+          figuration.

     POINTOPOINT
           Indicates that the address is a  point-to-point  link.
@@ -1043,8 +1051,8 @@
     dhcpinfo(1),  dhcpagent(1M),  in.mpathd(1M),  in.routed(1M),
     ndd(1M),  netstat(1M), ethers(3SOCKET), gethostbyname(3NSL),
     getnetbyname(3SOCKET), hosts(4),  netmasks(4),  networks(4),
-    hsswitch.conf(4),   attributes(5),   arp(7P),   ipsecah(7P),
-    ipsecesp(7P), tun(7M)
+    nsswitch.conf(4),    attributes(5),    zones(5),    arp(7P),
+    ipsecah(7P), ipsecesp(7P), tun(7M)

     System Administration Guide, Volume 3
```

D.2.2 poolbind(1M)

```
--- poolbind.1m.ref        Tue Apr  8 16:40:35 2003
+++ poolbind.1m.new        Tue Apr  8 16:40:35 2003
@@ -10,10 +10,10 @@
     /usr/sbin/poolbind -Q pid...

 DESCRIPTION
-    The poolbind command allows an authorized user to bind pro-
-    jects, tasks, and processes to pools. It can also allow a
-    user to query a process to determine which pool the process
-    is bound to.
+    The poolbind command allows an authorized user to bind
+    zones, projects, tasks, and processes to pools. It can also
+    allow a user to query a process to determine which pool the
+    process is bound to.

 OPTIONS
     The following options are supported:
@@ -43,6 +43,11 @@
                as either a project name or a numerical project
                ID. See project(4).

+          zoneid
+               idlist is a list of zone IDs. Bind all processes
+               within the list of zones to the specified pool.
+               Each zone ID can be specified as either a zone
+               name or a numerical zone ID. See zones(5).
     -q pid ...
          Queries the pool bindings for a given list of process
          IDs. If the collection of resources associated with
@@ -60,8 +65,8 @@
     The following operands are supported:

     poolname
-          The name of a pool to which the specified project,
-          tasks or processes are to be bound.
+          The name of a pool to which the specified zone, pro-
+          ject, tasks or processes are to be bound.

 EXAMPLES
```

```
     Example 1: Binding All Processes
@@ -118,7 +123,7 @@

 SEE ALSO
      pooladm(1M), poolcfg(1M), libpool(3LIB), project(4),  attri-
-     butes(5)
+     butes(5), zones(5)

      System Administration Guide: Resource Management and Network
      Services
```

D.2.3 prstat(1M)

```
--- prstat.1m.ref        Tue Apr  8 16:40:36 2003
+++ prstat.1m.new        Tue Apr  8 16:40:36 2003
@@ -2,20 +2,22 @@
      prstat - report active process statistics

 SYNOPSIS
-     prstat [-acJLmRtTv] [-C psrsetlist] [-j projlist] [-k  task-
+     prstat [-acJLmRtTvZ] [-C psrsetlist] [-j projlist] [-k task-
      list]  [-n ntop[,nbottom]] [-p pidlist] [-P cpulist] [-s key
-     | -S key ] [-u euidlist] [-U uidlist] [interval [count]]
+     | -S key ]  [-u  euidlist]  [-U  uidlist]  [-z  zoneidlist]
+     [interval [count]]

 DESCRIPTION
      The prstat utility iteratively examines all active processes
      on  the  system and reports statistics based on the selected
      output mode and sort order. prstat provides options to exam-
-     ine  only  processes matching specified PIDs, UIDs, CPU IDs,
-     and processor set IDs.
+     ine  only processes matching specified PIDs, UIDs, zone IDs,
+     CPU IDs, and processor set IDs.

-     The -j, -k, -C, -p, -P, -u, and -U options accept  lists  as
-     arguments. Items in a list can be either separated by commas
-     or enclosed in quotes and separated by commas or spaces.
+     The -j, -k, -C, -p, -P, -u, -U, and -z options accept  lists
+     as  arguments.  Items  in  a list can be either separated by
+     commas or enclosed in quotes  and  separated  by  commas  or
+     spaces.
```

```
     If you do not specify an option, prstat examines all
     processes and reports statistics sorted by CPU usage.
@@ -136,6 +137,15 @@
           received.  Statistics that are not reported are marked
           with the - sign.

+     -z zoneidlist
+           Report only processes or lwps whose zone ID is in  the
+           given  list. Each zone ID can be specified as either a
+           zone name or a numerical zone ID.  See zones(5).
+
+     -Z    Report information about processes and zones. In  this
+           mode  prstat displays separate reports about processes
+           and zones at the same time.
+
 OUTPUT
     The following list defines the column headings and the mean-
     ings of a prstat report:
@@ -282,7 +293,8 @@

 SEE ALSO
     proc(1),       psrinfo(1M),       psrset(1M),       sar(1M),
-    pset_getloadavg(3C), proc(4), project(4), attributes(5)
+    pset_getloadavg(3C),   proc(4),   project(4),   attributes(5),
+    zones(5)

 NOTES
     The snapshot of system usage displayed  by  prstat  is  true
```

D.3 System Calls

D.3.1 priocntl(2)

```
--- priocntl.2.ref        Mon Apr  7 10:10:45 2003
+++ priocntl.2.new        Mon Apr  7 10:10:46 2003
@@ -95,10 +95,14 @@
     P_UID The id argument is a user ID. The priocntl()  function
           applies to all LWPs with this effective user ID.

+     P_ZONEID
+           The id argument is a zone ID. The priocntl()  function
+           applies to all LWPs with this zone ID.
+
```

```
      An id value of P_MYID can be used in conjunction with the
      idtype value to specify the LWP ID, parent process ID, pro-
      cess group ID, session ID, task ID, class ID, user ID, group
-     ID, or project ID of the calling LWP.
+     ID, project ID, or zone ID of the calling LWP.

      To change the scheduling parameters of an LWP (using the
      PC_SETPARMS or PC_SETXPARMS command as explained below) ,
@@ -200,10 +205,10 @@
            Set or get nice value of the specified LWP(s) associ-
            ated with the specified process(es). When this command
            is used with the idtype of P_LWPID, it sets the nice
-           value of the LWP. The arg argument points to a struc-
-           ture of type pcnice_t. The pc_val member specifies the
-           nice value and the pc_op specifies the type of the
-           operation.
+           value of the LWP. The arg argument points to a
+           structure of type pcnice_t. The pc_val member speci-
+           fies the nice value and the pc_op specifies the type
+           of the operation.

            When pc_op is set to PC_GETNICE, priocntl() sets the
            pc_val to the highest priority (lowest numerical
@@ -304,9 +309,9 @@
            specific parameter data are described below and can
            also be found in the class-specific headers
            <sys/rtpriocntl.h>, <sys/tspriocntl.h>, and
-           <sys/fxpriocntl.h>. If the specified class is a con-
-           figured class and a single LWP belonging to that class
-           is specified by the idtype and id values or the
+           <sys/fxpriocntl.h>. If the specified class is a
+           configured class and a single LWP belonging to that
+           class is specified by the idtype and id values or the
            procset structure, then the scheduling parameters of
            that LWP are returned in the given (key, value) pair
            buffers. If the LWP specified does not exist or does
@@ -407,9 +412,9 @@
      The realtime class has a range of realtime priority (rt_pri)
      values that can be assigned to an LWP within the class.
      Realtime priorities range from 0 to x, where the value of x
-     is configurable and can be determined for a specific instal-
-     lation by using the priocntl() PC_GETCID or PC_GETCLINFO
-     command.
+     is configurable and can be determined for a specific
```

```
+     installation by using the priocntl() PC_GETCID or
+     PC_GETCLINFO command.

      The realtime scheduling policy is a fixed priority policy.
      The scheduling priority of a realtime LWP is never changed
```

D.4 Library Interfaces

D.4.1 getpriority(3C)

```
--- getpriority.3c.ref          Mon Apr  7 10:10:46 2003
+++ getpriority.3c.new          Mon Apr  7 10:10:46 2003
@@ -18,17 +18,17 @@
      Target processes are specified by the values of the which
      and who arguments.  The which argument may be one of the
      following values:  PRIO_PROCESS,  PRIO_PGRP,  PRIO_USER,
-     PRIO_GROUP,  PRIO_SESSION,  PRIO_LWP,  PRIO_TASK,  or
-     PRIO_PROJECT, indicating that the who argument is to be
+     PRIO_GROUP, PRIO_SESSION, PRIO_LWP, PRIO_TASK, PRIO_PROJECT,
+     or PRIO_ZONE, indicating that the who argument is to be
      interpreted as a process ID, a process group ID, an effec-
      tive user ID, an effective group ID, a session ID, an lwp
-     ID, a task ID, or a project ID, respectively.  A 0 value for
-     the who argument specifies the current process, process
-     group, or user. A 0 value for the who argument is treated as
-     valid group ID, session ID, lwp ID, task ID, or project ID.
-     A P_MYID value for the who argument can be used to specify
-     the current group, session, lwp, task, or project, respec-
-     tively.
+     ID, a task ID, a project ID, or a zone ID, respectively.  A
+     0 value for the who argument specifies the current process,
+     process group, or user. A 0 value for the who argument is
+     treated as valid group ID, session ID, lwp ID, task ID, pro-
+     ject ID, or zone ID. A P_MYID value for the who argument can
+     be used to specify the current group, session, lwp, task,
+     project, or zone respectively.

      If more than one process is specified, getpriority() returns
      the highest priority (lowest numerical value) pertaining to
@@ -71,7 +71,7 @@
           The value of the which argument was not recognized, or
           the value of the who argument is not a valid process
           ID, process group ID, user ID, group ID, session ID,
```

```
-           lwp ID, task ID, or project ID.
+           lwp ID, task ID, project ID, or zone ID.

      In addition, setpriority() may fail if:
```

D.4.2 ucred_get(3C)

```
--- ucred_get.3c.ref        Mon Apr  7 10:10:46 2003
+++ ucred_get.3c.new        Mon Apr  7 10:10:46 2003
@@ -1,8 +1,9 @@
 NAME
      ucred_get,   ucred_free,    ucred_geteuid,    ucred_getruid,
      ucred_getsuid,  ucred_getegid, ucred_getrgid, ucred_getsgid,
-     ucred_getgroups,       ucred_getpid,       ucred_getprivset,
-     ucred_getpflags - user credential functions
+     ucred_getgroups,       ucred_getpid,       ucred_getzoneid,
+     ucred_getprivset,  ucred_getpflags  -  user credential func-
+     tions

 SYNOPSIS
      #include <ucred.h>
@@ -30,6 +31,8 @@

      pid_t ucred_getpid(ucred_t *uc);

+     zoneid_t ucred_getzoneid(ucred_t *uc);
+
      uint_t ucred_getpflags(ucred_t *uc, uint_t flags);

      These functions return or act on a user credential, ucred_t.
@@ -61,6 +63,9 @@
      The ucred_getpid() function returns the process  ID  of  the
      process or -1 if the process ID is not available.

+     The ucred_getzoneid() function returns the zone  ID  of  the
+     process or -1 if the zone ID is not available.
+
      The  ucred_getprivset()  function  returns   the   specified
      privilege  set  specified  as  second  argument,  or NULL if
      either the requested information is  not  available  or  the
```

D.5 File Formats

D.5.1 proc(4)

```
--- proc.4.ref        Mon Apr  7 10:10:47 2003
+++ proc.4.new        Mon Apr  7 10:10:47 2003
@@ -256,6 +256,7 @@
           char pr_dmodel;          /* data model of the process */
           taskid_t pr_taskid;      /* task id */
           projid_t pr_projid;      /* project id */
+          zoneid_t pr_zoneid;      /* zone id */
           lwpstatus_t pr_lwp;      /* status of the representative lwp */
     } pstatus_t;

@@ -585,6 +586,7 @@
          lwpsinfo_t pr_lwp;       /* information for representative lwp */
          taskid_t pr_taskid;      /* task id */
          projid_t pr_projid;      /* project id */
+         zoneid_t pr_zoneid;      /* zone id */
     } psinfo_t;

     Some of the entries in psinfo, such as pr_flag and  pr_addr,
```

D.6 Protocols

D.6.1 if_tcp(7P)

```
--- if_tcp.7p.ref        Mon Apr  7 10:10:48 2003
+++ if_tcp.7p.new        Mon Apr  7 10:10:48 2003
@@ -60,6 +60,7 @@
           int                    lif_muxid[2];   /* mux id's for arp and ip */
           struct lif_nd_req      lifru_nd_req;
           struct lif_ifinfo_req  lifru_ifinfo_req;
+          zoneid_t               lifru_zone;     /* SIOC[GS]LIFZONE */
        } lifr_lifru;

     #define lifr_addrlen   lifr_lifru1.lifru_addrlen
@@ -80,6 +81,7 @@
     #define lifr_arp_muxid lifr_lifru.lif_muxid[1]
     #define lifr_nd        lifr_lifru.lifru_nd_req     /* SIOCLIF*ND */
     #define lifr_ifinfo    lifr_lifru.lifru_ifinfo_req /* SIOC[GS]LIFLNKINFO */
+    #define lifr_zone      lifr_lifru.lifru_zone       /* SIOC[GS]LIFZONE */
     };
```

```
     SIOCSLIFADDR
@@ -159,6 +159,12 @@
      SIOCSLIFINDEX
            Set the interface index associated with the interface.

+     SIOCGLIFZONE
+           Get the zone associated with the interface.
+
+     SIOCSLIFZONE
+           Set the zone associated with the interface.
+
      SIOCLIFADDIF
            Add a new logical interface on  a  physical  interface
            using an unused logical unit number.
@@ -205,9 +211,9 @@
            this node.

      SIOCTONLINK
-           Test  if  the  address  is  directly  reachable,  for
-           example, that it  can be reached without going through
-           a router. This request takes an sioc_addrreq structure
+           Test if the address is directly reachable,  for  exam-
+           ple,  that  it  can be reached without going through a
+           router. This request takes an  sioc_addrreq  structure
            (see  below)  as a value-result parameter. The sa_addr
            field should be set to the address to test. The sa_res
            field  will contain a non-zero value if the address is
```

D.7 Kernel Interfaces

D.7.1 ddi_cred(9F)

```
--- ddi_cred.9f.ref         Mon Apr  7 10:10:48 2003
+++ ddi_cred.9f.new         Mon Apr  7 10:10:48 2003
@@ -1,7 +1,7 @@
 NAME
      ddi_cred,   crgetuid,   crgetruid,   crgetsuid,    crgetgid,
-     crgetrgid, crgetsgid, crgetgroups, crgetngroups - access and
-     change parts of the cred_t structure
+     crgetrgid, crgetsgid, crgetzoneid, crgetgroups, crgetngroups
+     - access and change parts of the cred_t structure
```

```
SYNOPSIS
     #include <sys/cred.h>
@@ -18,6 +18,8 @@

     gid_t crgetsgid(const cred_t *cr);

+    zoneid_t crgetzoneid(const cred_t *cr);
+
     const gid_t *crgetgroups(const cred_t *cr);

     int crgetngroups(const cred_t *cr);
@@ -61,6 +63,9 @@
     tively,  the  effective,  real,  and saved group id from the
     user credential pointed to by cr.

+    crgetzoneid() returns the zone id from the  user  credential
+    pointed to by cr.
+
     crgetgroups() returns the group list of the user  credential
     pointed to by cr.
```

Andrew Tucker

Solaris Operating Environment

tucker@eng.sun.com

Jails: A Progress Report

Server Virtualization

- Customers want to deploy multiple services on same physical hardware
- Need vendor support to construct solutions
- Necessary attributes:
  - Resource controls
  - Security isolation
  - Fault containment

Possible Solutions

- Domains
- Virtual Machines/LPARs
- Resource Management
- Java Containers

Jails

- Originally implemented in FreeBSD 4.x
- Virtualizes OS layer: file system, devices, network
- Secure boundary around virtualized instance
- Provides:
  - Security: can't affect activity outside jail
  - Privacy: can't see outside jail
  - Fault isolation (unprivileged SW, some HW errors)
- Aligns well with resource management

Jails

Solaris

Goals

- "Interesting" network applications run unmodified in jail
  - Usual suspects: iAS, iWS, Apache, Zeus, pure Java, ...
  - Attack-prone network services: sendmail, BIND, ...
- Easy configuration and administration
  - Support large (full FS tree) or small (subdirectory)
  - Environment "not too different"
  - Align with resource management
- Performance at or close to non-jailed

Basic Principles

- Within a jail, all resources must be local, virtualized, or read-only (not exposing information from other jails)
- No escape from a jail (no chroot-like escapes)
- Entry to jail is controlled (no inherited fds, etc.)

Security

- Root can't be trusted
  - Most operations requiring root disabled in jail
  - Exceptions: file operations, set[ug]id, other local operations
- Add jail ID to process credential
  - *suser* and *drv_priv* fail for processes in jail
  - *suser_jail* succeeds if euid == 0
- May need to associate specific privileges with jails
  - Jail in separate processor set can call priocntl

File Systems

- chroot++ or mount context
  - Affects flexibility of configuration, not much else
- Some file systems changes needed
  - Virtualization: procfs, mntfs
  - Bug fixing: fdfs, lofs
- Need to allow mounts
  - No mounts of devices (possibility of FS corruption)
  - But unscrupulous user can corrupt mnttab
  - What does mnttab show for root device?

Devices

- Initial plan: no physical devices
  - Access through file system
  - Could also partition between jails – but be careful of shared HW (adapters, buses, etc.)
- Some pseudo devices also a problem
  - cpc, kmem, ...
- Need infrastructure support for managing devices, creating jail device tree
  - devfsadm, devinfo, specfs?, ...

Process Model

- Can't see or affect processes outside jail
  - Filtered through per-jail procfs mounts
- Jail associations visible through /proc, etc.
  - pkill -j *jailid*
- Issues:
  - init
  - ppid of jail "root" process
- How complete must this world be?

Name Service

- Can be completely localized
  - multiple copies of nscd, etc.
  - needed to support different administrative domains

    "Give customers their own root password"
- Could also be shared via nscd
  - mount /etc/.name_service_door in each jail
  - no local passwd, etc.

System V IPC

- Currently global ID space
  - Root can access/control any IPC
- Add jail ID to ipc_perm
  - Jail association established on create
  - Only accessible by processes in same jail
  - No cross-jail IPC (use the network or doors)
- Other possibilities: shmfs (or /dev/shm)
  - Allows cross-jail communication via namefs

Networking

- Need to be able to access the network
  - Individual ports or separate (per-jail) IP address
- No control of routing table, etc. within jail
- Local jail-jail communication should be possible (and fast?)
- Possible issues with loopback net, INADDR_ANY

Administration

- How do we make jails easy to administer?
- Configuration
  - Script support for constructing "standard" configs
  - Leverage diskless client tools — shared /usr, etc.
  - Fix lofs to enforce read-only mounts
  - Persistence, automatic instantiation, ...
- Management
  - Jail-aware utilities: prstat, ps, ...
  - Automatic binding to a jail: use projects?

Fault Isolation

- Currently E$ error in process data causes a reboot
  - Can't just kill process — what else may be depending on that process?
- Jails provide a natural fault isolation boundary
  - No communication (other than network) with other processes on machine
  - Kill (and restart?) jail on faults
- Isolation isn't total
  - Kernel panics
  - HW faults that can't be isolated to jail

Resource Management

- Complements security and fault isolation aspects of jails
- Make it easy to match jail & RM boundaries
  - Could bind to each in /etc/project - but what should settaskid() do?
- Don't constrain - want to allow flexibility
  - n jails ⇒ 1 pool
  - 1 jail ⇒ n pools
- What should psrinfo show inside pool/jail?

To Do List (Partial)

procfs
fdfs
suser/drv_priv
hasprocperm
shmsys, semsys, msgsys
tmpfs
namefs
ps
pgrep/pkill
prstat
mntfs
devinfo
devfsadm
lofs
ipcs

NFS?
autofs?
sockfs
other networking stuff (lots here)
init?
specfs?
pts/ptm/tty etc.
mount?
syslog?
utmp/wtmp?

Related Products

- FreeBSD
- Ensim
- Plan 9
- Trusted Solaris

Conclusions

- Jails (still) seem viable, useful
- Provide 2 of 3 virtualization attributes (RM provides the other)
- File system namespace is easy to partition
  - Flexible model for global ID mgmt (doors vs shm)
- Networking is (will be) the hard part
- Don't make administrator's job even harder
- Don't make apps change
- Lots of questions still to answer